Universidad Complutense de Madrid

Alga - Adaptative Loader for General Analytics.

Gabriel Emilio Lugo Estévez
Álvaro Fuentes Solas

Trabajo Fin de Grado

Ingeniería de Computadores

Facultad de informática

Dirigido por
Gonzalo Rubén Méndez Pozo
Javier Gómez Santos

Cargador Adaptable de datos para la realización de Análisis Generales.

Junio, 2021

Agradecimientos

En nombre de los dos integrantes de este proyecto deseamos dar las gracias en primer lugar a nuestros directores Javier Gómez Santos y Gonzalo Rubén Mendez Pozo por todo lo que hemos aprendido acerca de las arquitecturas Big Data y haber hecho muy interesante esta introducción al sector. También queremos mencionar a Rubén Villa Muñoz, ponente en una de las charlas de "La Semana De La Informática" en la Facultad de Informática y experto en Big Data en Everis. Queremos agradecerle su ayuda y su predisposición en ofrecernos su tiempo para el desarrollo de este proyecto junto con consejos y sugerencias.

Por otro lado Gabriel, agradece a Lucía, Orlando, Emilia y Mariarita por haberlo acompañado desde el primer día de la carrera y por haber celebrado cada uno de sus logros como si fueran los suyos propios.

Resumen

En este proyecto se realiza una arquitectura Big Data que permita adaptarse de la manera más sencilla posible a cualquier tipo de análisis de datos que se desee, con la posibilidad de aceptar las modificaciones necesarias para aplicar aprendizaje automático e identificar patrones, para realizar predicciones. Para este proyecto se utiliza un modelo de datos de la compañía The Phone House para el análisis de estos con el objetivo de tomar decisiones sobre los patrones en los datos identificados.

Palabras clave:

- Big Data
- Cloud Computing
- Lambda
- Azure
- Kafka
- PySpark
- Streaming
- Batch

Abstract

In this project we are going to develop a Big Data architecture that allows to adapt in the simplest possible way to any type of data analysis desired and with the possibility of accepting the modifications or aggregations that are needed. For this project, a data model of the telecommunications company The Phone House is used for the analysis of these data in order to improve customer campaigns.

Keywords:

- Big Data
- Cloud Computing
- Lambda
- Azure
- Kafka
- PySpark
- Streaming
- Batch

Índice general

Página

1. INTRODUCCIÓN **2**
1.1. Motivaciones . . . 3
1.2. Objetivos . . . 3
1.3. Metodología y plan de trabajo . . . 4
1.4. Estructura del documento . . . 7

2. INTRODUCTION **9**
2.1. Motivations . . . 10
2.2. Objectives . . . 10
2.3. Methodology and work plan . . . 11
2.4. Structure of the document . . . 13

3. ESTADO DEL ARTE **15**
3.1. Big Data . . . 15
3.1.1. Arquitecturas Big Data . . . 16
3.1.2. Extracción de datos . . . 17
3.1.3. Mensajería . . . 18
3.1.4. Almacenamiento . . . 19
3.1.5. Procesamiento . . . 21
3.1.6. Visualización . . . 26
3.2. Cloud Computing . . . 27
3.2.1. Fundamentos y tipos . . . 27
3.2.2. Problemas e Inconvenientes . . . 28
3.2.3. Futuro del Cloud Computing . . . 29

4. IMPLEMENTACIÓN **30**
4.1. Diseño y Estructura . . . 30
4.2. Software implementado . . . 32
4.2.1. Extracción de datos . . . 32
4.2.2. Colas de mensajería . . . 36
4.2.3. Capa Real Time . . . 39
4.2.4. Capa Batch . . . 43

5. RESULTADOS **48**
5.1. Visualización capa Real Time . . . 48
5.2. Explotación capa Batch . . . 51

6. CONCLUSIONES Y TRABAJO FUTURO **54**
6.1. Conclusiones 54
6.2. Trabajo futuro 56

7. CONCLUSIONS AND FUTURE WORK **57**
7.1. Conclusions 57
7.2. Future work 58

8. CONTRIBUCIONES **60**
8.1. Gabriel Emilio Lugo Estévez 60
8.1.1. Investigación 60
8.1.2. Prototipos 60
8.1.3. Implementación en la arquitectura 61
8.1.4. Memoria 61
8.1.5. Aportes extras 62
8.2. Álvaro Fuentes Solas 62
8.2.1. Investigación 62
8.2.2. Prototipos 63
8.2.3. Implementación en la arquitectura 63
8.2.4. Memoria 63

9. BIBLIOGRAFIA **65**

Capítulo 1

INTRODUCCIÓN

La evolución del internet en los últimos años ha provocado el paso de una red de investigación a una red de comunicaciones global. Con la llegada de los dispositivos móviles inteligentes, se ha incrementado a gran escala el desarrollo de servicios y aplicaciones que satisfacen las necesidades de millones de usuario que se suman a la utilización de estos dispositivos para mantenerse conectados de manera global. El uso de un gran conjunto de servicios provoca cantidades inimaginables de datos por parte de los usuarios. Entre ellos podemos mencionar desde los datos personales, hasta los datos obtenidos por utilización y rastreo de las propias aplicaciones. Todos estos conjuntos de datos son considerados el petróleo de la era digital, debido al gran potencial financiero que estos representan. ((Klein, Tran-Gia, and Hartmann, 2013))

Aunque el nombre de Big Data hace referencia a un gran conjunto de datos, estas tecnologías no solo toman en cuenta el volumen de los datos. Los ecosistemas Big Data manejan un concepto denominado "3 V": volumen, variabilidad y velocidad. El Big Data es todo aquello que tiene que ver con grandes volúmenes de información que se procesan y analizan con alta velocidad, presentando una variabilidad en su composición. ((Tascón, 2013)).

Los datos tienen una gran importancia en la actualidad para las empresas, cada vez más necesitan de sistemas complejos que implementen los mecanismos necesarios para explotar toda esa información que es recabada día tras día. La necesidad de conseguir datos llega hasta tal punto, que una gran cantidad de aplicaciones ofrecen servicios gratuitos, beneficiándose de la captación de los datos de los usuarios y de su uso posterior.

La existencia del Big Data en una empresa permite el análisis de todos estos grandes conjuntos de datos, basándose en el desarrollo de tareas como la inteligencia artificial o la creación de indicadores de rendimiento para obtener patrones, entre otras actividades que proporcionan un gran valor para el desarrollo y la evolución de la compañía.

El propósito de adentrarnos en la investigación y la implementación de una arquitectura Big Data compleja, se debe al deseo de presentar soluciones a una serie de necesidades que pueda presentar una empresa, con esto se quiere seguir desarrollando y evolucionando con dirección a la digitalización. Nos propusimos desarrollar una arquitectura que permita la realización de análisis en los datos para la toma de decisiones. Con esto queremos ofrecer un ecosistema que abarque todos los puntos esenciales del Big Data, que van desde la

extracción de la información, la estructuración y el paso de la información, con su debido almacenamiento y la explotación de esta información. Con esto también se quieren alistar los cimientos para la aplicación de aprendizaje automático sobre las decisiones, entre otras variantes que el Big Data y la inteligencia artificial nos proponen.

1.1. Motivaciones

En este proyecto se trabaja con la empresa The Phone House S.L.U. Esta compañía es una de las principales en el ámbito de las ventas de servicios y tecnología que consta de un departamento de datos en el cual participan distintos analistas e ingenieros que se encargan del manejo de los mismos. Es aquí donde surge la necesidad de obtener una arquitectura Big Data que les permita trabajar con una parte de esa información que obtienen de sus distintos sistemas y obtener unos resultados concretos y de la manera más eficiente.

La empresa genera numerosas campañas de promociones ofrecidas a clientes, pero debido al gran auge de la digitalización en empresas tecnológicas y la competencia directa con otras compañías del sector dedicadas única y exclusivamente a venta de servicios, surge la necesidad de generar promociones enfocadas a las necesidades de los clientes y enfocadas principalmente a clientes potenciales. Con esto aparte de seguir fidelizando a los clientes, se quiere lograr un incremento en las ventas cruzadas de servicios.

Actualmente se encuentra la promoción de cumpleaños como campaña particular a un cliente. Con esta implementación se quiere analizar variables que permitan la aparición de nuevas campañas particulares de cara al cliente y con ello conseguir cierre y cruce de ventas basado en los intereses de la empresa.

Partiendo de esta necesidad, nos surge la oportunidad de desarrollar un proyecto aplicado a necesidades reales, con el objetivo de conseguir mejoras notables en la evolución de la empresa. Es por esto, que tiene sentido profundizar en las tecnologías y las herramientas utilizadas en entornos Big Data, para crear una arquitectura completa y escalable, que nos permita tratar la información de las transacciones realizadas en los sistemas de facturación de la empresa y a su vez la captación de los nuevos clientes. Al estar hablando de una empresa de esta envergadura con grandes lotes de datos almacenados a lo largo del tiempo, debemos realizar a su vez una serie de procesos que sean eficientes para el tratamiento de la misma.

La empresa cuenta con un proceso de digitalización. Por ello, es importante que los nuevos desarrollos e implementaciones sigan esta premisa de aplicación en entornos en la nube, para proporcionar más escalabilidad y permitir sin mucha complejidad, la realización de integraciones con sus sistemas internos.

1.2. Objetivos

El objetivo principal es la implementación de una arquitectura Big Data completa que permita almacenar y analizar cualquier conjunto de datos disponibles y con esto explotar indicadores de valor de negocio extraídos de dichos datos. Se busca una arquitectura que sea lo más moldeable y escalable factible para su posible aumento de alcance en un futuro, incorporando nuevas funcionalidades de explotación de los datos como el aprendizaje

automático (*Machine learning*) o su posible incorporación en otros proyectos tanto en su totalidad como partes independientes de esta.

Esta arquitectura debe constar de un proceso de extracción de los datos, análisis y procesamiento de los mismos junto con su almacenamiento y su posterior visualización. Los casos de uso concretos cubiertos por este proyecto buscan cubrir la necesidad de la creación de nuevas campañas de promociones para los clientes según los resultados que se obtengan por esta arquitectura entre otras posibles opciones.

Para cubrir estas necesidades, planteamos una serie de objetivos de menor tamaño para llevar a cabo dentro de la propia arquitectura, siguiendo los requisitos indicados por parte de la empresa. Se pueden mencionar los siguientes:

- Implementar una arquitectura Big Data de dos capas, para el tratamiento de información en tiempo real y procesos de análisis de grandes conjuntos de datos en periodos de tiempo determinados.
- Integrar un *data lake* para almacenar toda la información de la empresa que se va utilizando paralelamente a los procesos de análisis y sin sufrir ningún tipo de modificación.
- Desarrollar un generador de datos que sustituya la información sensible de la empresa y de los clientes alojada en la base de datos de la compañía en BigQuery.
- Acceso controlado a la información almacenada en los repositorios, mediante *API REST*.
- Integración de colas de mensajería para el traspaso de la información.
- Implementación de una arquitectura sobre repositorios alojados en entornos Cloud.
- Utilización de tecnologías escalables que permitan adaptar de una manera sencilla cualquier estructura de datos para su análisis posterior.
- Utilización de herramientas visuales para la explotación de la información en tiempo real.

1.3. Metodología y plan de trabajo

En esta sección damos paso a explicar la metodología y el plan utilizado para el desarrollo total del proyecto.

En primera instancia analizaremos los requisitos propuestos por la empresa, basándonos en los objetivos a conseguir. A continuación realizaremos el diseño de la arquitectura y los componentes necesarios para la construcción total del sistema Big Data. Posteriormente damos paso a realizar los diseños de los distintos programas y aplicaciones que intervienen en cada etapa del sistema.

Una vez concretados los diseños damos paso a la codificación de los programas para posteriormente finalizar con la realización de las pruebas del mismo.

Para el control de versionado de nuestro código utilizaremos un repositorio privado en

(https://github.com/AlgaBigData)

Estructuramos el plan de trabajo en seis etapas principales:

- Definición del ámbito al que se aplica: Basados en el planteamiento de implementar una arquitectura Big Data, empezamos por definir el ámbito en el que se aplicará, para esto llevamos a cabo el cierre de la idea.

  En esta etapa realizamos la búsqueda de los orígenes de datos con los que trabajaremos. Luego realizaremos un pequeño programa en el que podamos conectarnos a la fuente de datos, para extraerlos y representarlos de una manera básica.

- Estudio de colas de mensajería: Una vez desarrollado el primer programa básico y con una idea clara del diseño que queremos implementar, procedemos al estudio de las distintas herramientas para el envío de datos a través de colas de mensajería. Para ello, realizamos una investigación detallada del funcionamiento y compatibilidad con nuestro proyecto, comparando las ventajas y desventajas de cada una de las colas. Es importante recalcar que la elección de las colas es un paso fundamental en la arquitectura ya que estas representan los canales de paso de los distintos flujos de datos con los que trabajaremos.

  Luego de seleccionar la cola, debemos adaptar el programa realizado en la primera etapa, para guardar la información que obtenemos de las fuentes en la cola de mensajería seleccionada.

- Volcado de mensajes al repositorio de datos: Al tener los datos introducidos en la cola de mensajería, debemos realizar un proceso de volcado de datos, en el cual debemos desarrollar un programa que los extraiga de la cola de mensajería y se encargue de volcarlos en un almacén de datos.

  Para la elección de los repositorios de datos a utilizar, realizamos un estudio buscando principalmente herramientas que nos proporcionan escalabilidad y facilidad de uso. Para esto proponemos estudiar distintos repositorios en la nube como opciones principales.

- Explotación de datos de los repositorios: Una vez poblado nuestros repositorios de datos, la idea es desarrollar un programa que nos permita realizar consultas de los datos de manera eficiente y con poca interacción con el usuario. Este programa se encargará de realizar tareas repetitivas y debe ser capaz de paralelizar el procesamiento de grandes cantidades de datos en distintos *clusters* para su explotación.

  En el procesamiento por lotes se les debe dar significado a los datos a un nivel entendible por parte de negocio y los explotaremos con la realización de informes para su debido análisis.

- Procesamiento de datos en tiempo real: Para llevar a cabo una etapa tan fundamental en una arquitectura Big Data como es el procesamiento de los datos en tiempo real, debemos realizar un estudio de las distintas tecnologías que nos permiten realizar este procesamiento, entre ellas escogeremos la que más se adapte a nuestras necesidades basándonos en un análisis profundo de los que nos ofrece cada una de las tecnologías principales y más utilizadas actualmente.

  La importancia de estudiar las más utilizadas recae en que a la hora de obtener

documentación, conseguimos mucha investigación por parte de usuarios y a su vez contamos con soporte por parte de la comunidad, esto nos da mucha ventaja a la hora de corregir posibles errores o bloqueos que nos aparezcan en el camino.

El procesamiento en tiempo real se realizará con un programa que esté a la espera de nuevos datos, consultando directamente de la cola de mensajería. A diferencia de la etapa anterior, el procesamiento se hará con los datos que van siendo introducidos en tiempo real y se van procesando y dando significado a nivel de entendimiento por parte de negocio a través de plazos cortos de tiempo.

Finalmente se almacenarán los datos procesados en repositorios que puedan ser consultados posteriormente para su explotación.

- Explotación de datos en tiempo real: Como etapa final mostramos los datos anteriormente procesados. Para llevar a cabo esta tarea comparamos de nuevo distintas herramientas de visualización para consultar los repositorios con los datos agrupados y previamente procesados ofreciendo una visualización en tiempo real de todos los nuevos datos que van entrando a nuestra plataforma. Esta visualización requiere de hacerlo con el fin de llevar a cabo análisis de los mismos para la toma de decisiones a niveles empresariales.

La planificación general del proyecto se realizó teniendo en cuenta el rango desde Octubre de 2020 a Junio 2021. En la siguiente tabla mostramos los objetivos propuesto desde un inicio y como se van modificando con el paso de los meses.

Periodo	Actividades
Octubre	Primera toma de contacto, cierre de la idea y presentación del grupo. Inicio de investigación de conexiones a fuentes de datos. Primera reunión con la empresa para la toma de requisitos.
Noviembre	Desarrollo de programas que extraen datos, familiarización con Python con pruebas y demostraciones a los tutores de su funcionamiento. Inicio de investigación de las colas de mensajería.
Diciembre	Reunión con la empresa para aclaración de dudas de implementación y solicitud de credenciales de acceso a las fuentes y la estructura de los datos. Comienzo con el desarrollo de los generadores de datos con las características indicadas por la empresa junto con su inserción a las colas de mensajería.
Enero	Investigación de los repositorios de datos, investigación de las tecnologías de procesamiento de datos en tiempo real, creación de cuentas en los repositorios Cloud de Azure
Febrero	Realización de pruebas para comparación de los repositorios. Desarrollo de programas que conectan con los repositorios de datos. Desarrollo de programas consumidores de colas de mensajería y volcado de datos en los repositorios.
Marzo	Taller de "Aspectos formales para presentar un trabajo académico" impartido en la Universidad Complutense. Investigación de las herramientas de visualización. Inicio en la redacción de borrador de la memoria.
Abril	Taller de "Arquitecturas Lambda sobre Azure" impartido por Rubén Villa y Carlos Espilez en la Universidad Complutense. Implementación de arquitectura lambda en nuestro proyecto. Pruebas en la arquitectura Big data construida.
Mayo	Procesamiento y explotación de los datos en tiempo real. Procesamiento *Batch*. Implementación de programas que analizan los datos por lotes.
Junio	Entrega del borrador de la memoria. Entrega final de la memoria. Defensa y demostración del trabajo final.

1.4. Estructura del documento

En el capítulo 3 se hace una explicación sobre los campos en los que se ha trabajado en este proyecto. En primer lugar, en la sección 3.1 hacemos una introducción al Big Data, hablamos de la importancia adquirida a lo largo de los años, porque está siendo tan relevante en el sector empresarial y los puntos más trascendentes a tener en cuenta cuando hablamos de este término. Por otro lado, en la sección 3.2, desarrollamos un pilar fundamental en el Big Data como es el Cloud Computing y en el que se basa

principalmente la implementación de este proyecto. Explicamos, qué es y en que se basa el desarrollo en la nube, junto con sus problemas y los puntos a mejorar.

En el capítulo 4 desarrollamos el trabajo llevado a cabo en este proyecto. Por un lado, en la sección 4.1 se detallan los fundamentos de esta arquitectura junto con la estructura y las distintas partes en las que esta dividida. En la sección 4.2 se explican las aplicaciones software implementadas, los motivos de su elección frente a otros posibles modelos, las conexiones entre las distintas aplicaciones y los lenguajes utilizados para ello.

En el capítulo 5 se muestran los resultados obtenidos tanto de los informes generados por un lado como la representación visual por otro.

En el capítulo 6 detallamos nuestras conclusiones y trabajo futuro. En la sección 6.1 explicamos nuestra experiencia trabajando en entornos Big Data y Cloud Computing así como los inconvenientes que nos hemos encontrado y cómo los hemos solventado. En la sección 6.2, hacemos un resumen de cómo se podría ampliar esta arquitectura junto con la implementación de posibles mejoras.

Capítulo 2

INTRODUCTION

The evolution of the Internet in recent years has led to the transition from a research network to a global communications network. With the advent of smart mobile devices, the development of services and applications that meet the needs of millions of users who use these devices to stay connected globally has increased on a large scale. The use of a large set of services causes unimaginable amounts of data on the part of users. Among them we can mention from personal data, to data obtained from the use and tracking of the applications themselves. All these data sets are considered the oil of the digital age, due to the great financial potential they represent. ((Klein, Tran-Gia, and Hartmann, 2013))

Although the name Big Data refers to a large data set, these technologies do not only take into account the volume of data. Big Data ecosystems manage a concept called "3 V": volume, variability and velocity. Big data is everything that has to do with large volumes of information that are processed and analyzed at high speed, presenting variability in its composition. ((Tascón, 2013)).

Data is currently of great importance for companies, which increasingly need complex systems that implement the necessary mechanisms to exploit all the information that is collected day after day. The need to obtain data has reached such an extent that a large number of applications offer free services, benefiting from the collection of user data and its subsequent use.

The existence of Big Data in a company allows the analysis of all these large data sets, based on the development of tasks such as artificial intelligence or the creation of performance indicators to obtain patterns, among other activities that provide great value for the development and evolution of the company.

The purpose of getting into the research and implementation of a complex Big Data architecture, is due to the desire to present solutions to a series of needs that a company may present, with this we want to continue developing and evolving in the direction of digitization. We set out to develop an architecture that allows the realization of data analysis for decision making. With this we want to offer an ecosystem that covers all the essential points of Big Data, ranging from the extraction of information, structuring and passing of information, with its proper storage and exploitation of this information. With this, we also want to lay the foundations for the application of automatic learning

on decisions, among other variants that Big Data and artificial intelligence propose to us.

2.1. Motivations

In this project we work with the company The Phone House S.L.U. This company is one of the main in the field of sales of services and technology that has a data department in which different analysts and engineers who are responsible for handling them participate. It is here where the need arises to obtain a Big Data architecture that allows them to work with some of the information obtained from their different systems and obtain specific results in the most efficient way.

The company generates numerous promotional campaigns offered to customers, but due to the boom of digitization in technology companies and direct competition with other companies in the sector dedicated solely and exclusively to selling services, the need arises to generate promotions focused on the needs of customers and focused primarily on potential customers. With this, apart from continuing to build customer loyalty, we want to achieve an increase in cross-sales of services.

Currently there is a birthday promotion as a particular campaign to a client. With this implementation we want to analyze variables that allow the emergence of new particular campaigns for the customer and thus achieve closure and cross-selling based on the interests of the company.

Based on this need, we have the opportunity to develop a project applied to real needs, with the aim of achieving significant improvements in the evolution of the company. This is why it makes sense to deepen in the technologies and tools used in Big Data environments, to create a complete and scalable architecture, which allows us to treat the information of the transactions made in the billing systems of the company and in turn the attraction of new customers. As we are talking about a company of this size with large batches of data stored over time, we must in turn perform a series of processes that are efficient for the treatment of the same.

The company has a digitalization process. Therefore, it is important that new developments and implementations follow this premise of application in cloud environments, to provide more scalability and allow without much complexity, the realization of integrations with their internal systems.

2.2. Objectives

The main objective is the implementation of a complete Big Data architecture that allows storing and analyzing any available data set and thus exploiting business value indicators extracted from such data. We are looking for an architecture that is as moldable and scalable as possible for its possible increase in scope in the future, incorporating new data exploitation functionalities such as machine learning or its possible incorporation in other projects both as a whole and as independent parts of it.

This architecture should consist of a process of data extraction, analysis and processing of the data together with its storage and subsequent visualization. The specific use cases

covered by this project seek to cover the need for the creation of new promotional campaigns for customers according to the results obtained by this architecture among other possible options.

To cover these needs, we propose a series of smaller objectives to be carried out within the architecture itself, following the requirements indicated by the company. The following can be mentioned:

- Implement a two-layer Big Data architecture, for Real Time information processing and analysis processes of large data sets in specific periods of time.
- Integrate a data lake to store all the company's information that is used in parallel to the analysis processes and without undergoing any type of modification.
- Develop a data generator that replaces sensitive company and customer information stored in the company's database in BigQuery.
- Controlled access to the information stored in the repositories, through REST APIs.
- Integration of messaging queues for information transfer.
- Implementation of an architecture on repositories hosted in Cloud environments.
- Use of scalable technologies that allow to easily adapt any data structure for further analysis.
- Use of visual tools to exploit information in Real Time.

2.3. Methodology and work plan

In this chapter we will explain the methodology and the plan used for the total development of the project.

First we will analyze the requirements proposed by the company, based on the objectives to be achieved. Then we will design the architecture and the necessary components for the total construction of the Big Data system. Afterwards, we will design the different programs and applications involved in each stage of the system.

Once the designs have been finalized, we move on to the coding of the programs and then finish with the testing of the system.

For the versioning control of our code we will use a private repository in Github

(https://github.com/AlgaBigData)

We structured the work plan in six main stages

- Definition of the scope to which it applies: Based on the approach to implement a Big Data architecture, we begin by defining the scope in which it will be applied, for this we carry out the closure of the idea.

  At this stage we carry out the search for the data sources with which we will work. Then we will make a small program in which we can connect to the data source, to extract and represent them in a basic way.

- Study of messaging queues: Once the first basic program has been developed and with a clear idea of the design we want to implement, we proceed to the study of the different tools for sending data through messaging queues. To do this, we conducted a detailed investigation of the operation and compatibility with our project, comparing the advantages and disadvantages of each of the queues. It is important to emphasize that the choice of the queues is a fundamental step in the architecture since they represent the channels of passage of the different data flows with which we will work.

  After selecting the queue, we have to adapt the program we created in the first step to store the information we obtain from the sources in the selected messaging queue.

- Dump messages to the data repository: Having the data entered in the messaging queue, we must perform a data dump process, in which we must develop a program that extracts them from the messaging queue and is responsible for dumping them into a data repository.

  In order to choose the data repositories to be used, we carried out a study mainly looking for tools that provide scalability and ease of use. For this we propose to study different cloud repositories as main options.

- Data exploitation of the repositories: Once our data repositories are populated, the idea is to develop a program that allows us to perform data queries efficiently and with little interaction with the user. This program will be in charge of performing repetitive tasks and must be able to parallelize the processing of large amounts of data in different clusters for its exploitation.

  In Batch processing, the data must be given meaning at a level understandable by the business and exploited with reports for analysis.

- Real Time data processing: To carry out such a fundamental stage in a Big Data architecture as Real Time data processing, we must carry out a study of the different technologies that allow us to perform this processing, among them we will choose the one that best suits our needs based on a thorough analysis of what each of the main and most currently used technologies offers us.

  The importance of studying the most used ones lies in the fact that at the time of obtaining documentation, we get a lot of research by users and at the same time we have support from the community, this gives us a lot of advantage when correcting possible errors or blockages that appear on the way.

  Real Time processing will be done with a program that is waiting for new data, querying directly from the messaging queue. Unlike the previous stage, the processing will be done with the data that are being introduced in Real Time and are being processed and giving meaning at the level of understanding by the business through short periods of time.

  Finally, the processed data will be stored in repositories that can be consulted later for its exploitation.

- Real Time data exploitation: As a final stage we display the previously processed data. To carry out this task we again compare different visualization tools to consult

the repositories with the grouped and previously processed data, offering a Real Time visualization of all the new data entering our platform. This visualization needs to be done in order to carry out analysis of the data for decision making at enterprise levels.

The general planning of the project was carried out taking into account the range from October 2020 to June 2021. The following table shows the objectives proposed from the beginning and how they are modified as the months go by.

Month	Work
October	First contact, closing of the idea and presentation of the group. Start of research of connections to data sources. First meeting with the company to take requirements.
Noviembre	Development of programs that extract data, familiarization with Python with tests and demonstrations to tutors of its operation. Start of research on messaging queues.
December	Meeting with the company for clarification of implementation doubts and request of access credentials to sources and data structure. Start with the development of the data generators with the characteristics indicated by the company together with their insertion to the messaging queues.
January	Researching data repositories, researching Real Time data processing technologies, creating accounts in Azure Cloud repositories.
February	Performing tests for comparison of the repositories. Development of programs that connect with the data repositories. Development of programs that consume messaging queues and dump data in the repositories.
March	
April	Workshop on "Formal aspects to present an academic paper.[a]t the Complutense University of Madrid. Investigation of visualization tools. Beginning of the drafting of the report.
May	Real Time data processing and exploitation. Batch processing. Implementation of programs that analyze data in batches.
June	Delivery of the draft of the report. Final delivery of the report. Defense and demonstration of the final work.

2.4. Structure of the document

In chapter 2 we explain the fields in which we have worked in this project. First of all, in section 2.1 we make an introduction to Big Data, we talk about the importance acquired over the years, why it is being so relevant in the business sector and the most important points to consider when we talk about this term. On the other hand, in section 2.2, we develop a fundamental pillar in Big Data such as Cloud Computing and on which the implementation of this project is mainly based. We explain, what it is and what cloud development is based on, along with its problems and points to improve.

In chapter 3 we develop the work carried out in this project. On the one hand, section 3.1

details the fundamentals of this architecture along with the structure and the different parts into which it is divided. Section 3.2 explains the software applications implemented, the reasons for their choice over other possible models, the connections between the different applications and the languages used for this purpose. In addition, we also show the results obtained from the reports generated on the one hand and the visual representation on the other hand.

In chapter 4 we detail our conclusions and future work. In section 4.1 we explain our experience working in Big Data and Cloud Computing environments as well as the drawbacks we have encountered and how we have solved them. In section 4.2, we summarize how this architecture could be extended along with the implementation of possible improvements.

Capítulo 3

ESTADO DEL ARTE

En este apartado se realiza una breve descripción del tema sobre el que trata este proyecto, el Big Data. Se hace una breve definición de lo que es y en qué consiste el Big data junto con todo lo que abarca este término y la explicación de muchas de las herramientas y tecnologías utilizadas en este sector y por qué. Por otro lado, se explica en qué consiste el Cloud Computing, como influye y porque es una parte importante del Big Data. Indicamos sus utilidades, beneficios y cómo va a ir evolucionando y haciendo evolucionar al Big Data en un futuro.

3.1. Big Data

Se denomina Big Data al proceso de análisis y extracción de grandes cantidades de información con el fin de obtener una mayor comprensión de ellos. El Big Data tiene en cuenta el volumen, la velocidad, la variedad, la veracidad y el valor de los datos a analizar para garantizar un procesamiento eficiente y la satisfacción en los resultados. Entre muchas otras definiciones podríamos decir que el Big Data, es una cantidad gigante de datos que no puede ser tratada por programas convencionales, debido al volumen de los datos que esta representa, la velocidad de procesamiento y almacenamiento.

El gestionar datos pesados y realizar operaciones complejas con estos datos implica utilizar herramientas y técnicas capaces de soportarlo. Estas herramientas y técnicas conforman el ecosistema de Big data y su esfera.

Normalmente en los ecosistemas Big data se obtiene la información de múltiples fuentes, la cual necesita de un tratamiento minucioso que les otorgue sentido y coherencia.

Lo mismo sucede con las herramientas para el guardado de los datos. Los datos involucrados en el Big Data pueden ser estructurados o no estructurados, naturales o procesados y pueden o no, estar relacionados con el tiempo. Esta información normalmente es almacenada de manera distribuida en repositorios no relacionales. Un uso común son los *data lakes*, que permiten almacenar y procesar datos sin esquema y en cualquier formato sin la necesidad de conocer cómo se van a explotar en el futuro. Esta característica evita que sean necesarios complejos procesos de limpieza y preparación de dicha información.

La ventaja de almacenar los datos de manera distribuida, es la velocidad de procesamiento que esta permite para tratar dicha información, debido a que se le puede asignar a distintos

nodos la tarea de lectura y tratamiento de esos datos.

3.1.1. Arquitecturas Big Data

No existe una solución genérica para todos los casos de uso a los que se puede aplicar el Big Data. Cada proceso de análisis en donde se desea implementar el Big Data tiene que ser elaborado y construido de manera efectiva según los requisitos y objetivos fijados previamente. Es por ello necesario hacer uso de algunas de las diferentes arquitecturas Big Data, ya que la combinación de varias tecnologías dará lugar al caso de uso resultante. Al establecer una arquitectura fija se puede garantizar que se proporcionará una solución viable para el caso de uso solicitado ((Ozgur, Kleckner, and Li, 2015)).

Arquitectura Data Streaming Esta arquitectura procesa los datos de forma continuada según se van recibiendo. Los datos procesados se envían directamente a los consumidores en forma de mensajes. Aunque hay una parte de almacenamiento, éste se almacena más en forma de ventanas, por lo que el almacenamiento no se produce en un *data lake*, sino en el sistema periférico consiguiendo que la efectividad de los datos sea muy alta aunque la repetición de datos y las estadísticas históricas no pueden ser bien soportadas.

Arquitectura Lambda La mayoría de las arquitecturas son básicamente arquitectura Lambda o arquitecturas basadas en sus variantes. El objetivo de estas arquitecturas es satisfacer las necesidades de una amplia gama de procesos y casos de uso en el que se requieren lecturas y actualizaciones de baja latencia y evitando cualquier tipo de fallo, tanto de *hardware* como humano ((Hausenblas, 2015)).

El canal de datos de Lambda se divide en dos ramas: *Real Time* y *Batch*. El análisis en tiempo real depende básicamente de la arquitectura de *Real Time* para garantizar la eficacia de este procesamiento, el cálculo incremental es la principal referencia auxiliar, mientras que la capa *Batch* se basa en un procesamiento por lotes de información garantizando un análisis más consistente y detallado. Esto proporciona cubrir muy bien los escenarios de análisis de datos aunque suponga que la capa *Batch* y el flujo en tiempo real se enfrenten a diferentes escenarios, y la implementación se base en el uso de gran variedad de procesos y módulos distintos.

Arquitectura Kappa La arquitectura Kappa está optimizada sobre la base de Lambda, pero con el sistema de procesamiento por lotes eliminado. Para reemplazar este procesamiento, los datos simplemente se alimentan a través del sistema de *streaming*. En la arquitectura Kappa una capa introduce los datos desde el almacén de registros a un sistema de cálculo de flujo y otra proporciona respuestas optimizadas a las consultas. Pathirage ((2017)).

Esta arquitectura utiliza el procesamiento de flujos como pilar, pero los datos se almacenan en el *data lake*. Cuando se requieren análisis masivos o cálculos múltiples, los datos del *data lake* pueden transferirse de nuevo a través de la cola de mensajes consiguiendo resolver la parte redundante de la arquitectura Lambda y con la posibilidad de la repetición de datos aunque convierta a esta arquitectura en difícil de implementar.

Arquitectura Unifield Todas las arquitecturas anteriores se centran principalmente en el procesamiento masivo de datos, mientras que la arquitectura Unifield combina

el aprendizaje automático y el procesamiento de datos. En el núcleo, Unifield sigue basándose en Lambda, pero se ha transformado en el procesamiento de flujos. A esta capa se le ha añadido una nueva capa de *machine learning*. En los centros de datos y *data lakes* a través del canal de datos, se añade una nueva parte de *machine learning* y se utiliza en la capa de *streaming* combinando los análisis de ambos procesos. Esto supone un aumento de la complejidad afectando tanto a la implementación software como la implementación de hardware. Hussain ((2018))

3.1.2. Extracción de datos

El inicio de cualquier arquitectura Big Data que se desee implementar se basa en la extracción de datos. Los datos son el pilar fundamental y por lo que las fuentes de donde se obtienen los datos y la manera en la que son extraídos tiene un valor muy importante en la arquitectura pese a que en pasos posteriores nos preocuparemos por darle forma a estos datos y proporcionarle cierto significado.

Los datos se encuentran en todas partes, los podemos encontrar estructurados, semi-estructurados o no estructurados lo que varía dependiendo de las fuentes que los proporcionan. Algunas de ellas ofrecen metodologías de acceso bastante simples y documentadas, pero en muchas otras se tienen que aplicar técnicas bastante complejas para extraerlos.

Siguiendo esta premisa podemos mencionar algunas maneras más comunes a la hora de extraer datos.

Datasets Extraer información de *datasets* es una práctica muy común cuando hablamos de Big Data. Estos representan un conjunto de datos estructurados en un sistema de almacenamiento. Se pueden encontrar en múltiples formatos, uno de los más comunes es en formato CSV y se representan de manera tabulada. Un archivo CSV(*Comma-separated values*) tiene un formato de valores separados por comas que permite que los datos se guarden en un formato con estructura de tabla. ((Ads))

Este tipo de conjunto de datos suele estar disponible en páginas gubernamentales, empresas que proporcionan parte de sus datos no sensibles de manera pública o redes sociales, entre otras cantidades indefinidas de fuentes.

API Una API (*Application programming interface*) es un conjunto de definiciones y protocolos que se utilizan para integrar aplicaciones permitiendo que distintos servicios de una aplicación se puedan comunicar entre ellos sin la necesidad de conocer cómo se estructuran internamente ((Red Hat, 2021)). Este método proporciona documentación para que el usuario pueda realizar una serie de llamadas a los servicios disponibles, si se cumple dicha estructura se obtiene una respuesta con los datos solicitados.

Hoy en día podemos encontrar una gran cantidad de plataformas que nos proporcionan este tipo de acceso a sus datos aunque como requisito previo debemos crear cuentas de usuario para obtener credenciales y poder realizar solicitudes a dichas interfaces. Una vez hecho esto podemos recopilar gran cantidad de datos estructurados en formatos estándar.

Una ventaja que obtenemos al utilizar esta metodología es ahorrarnos todo el proceso de desarrollo para consultar cualquier servicio interno de una plataforma, simplemente se

deben realizar las peticiones basándonos en una estructura fija y obtenemos las respuestas. También es importante destacar que nos limitamos a obtener solo la información que está disponible en dichos servicios.

Web Scraping Es una variante que se puede utilizar en páginas que no proporcionan los medios mencionados anteriormente. En los casos en los que se necesita extraer datos y no se tiene disponibilidad de ninguna forma, aparece esta alternativa. Esta técnica consiste en utilizar un *software* que simula la interacción del usuario en las páginas web, recopilando la información que se encuentra en cualquier página que esté desplegada en un servidor. A continuación, la información pasa por un proceso de estructuración de la información recogida. La legalidad de realizar esta práctica, depende del país en el que se realice y de los términos y condiciones de la página en la que se recopila la información.

3.1.3. Mensajería

Para el envío de la información obtenida de las fuentes de datos a los distintos servidores de almacenamiento, existe una gran variedad de herramientas dependiendo del tipo de información que se va a enviar, la estructura de esta o el proceso al que va a ser sometida.

Para este proceso de envío de la información hemos seleccionado la herramienta Apache Kafka. Es una plataforma basada en la computación distribuida que se utiliza para procesar datos en tiempo real y se caracteriza por ser un servicio de mensajería escalable en lugar de tener un *broker* de mensajes individual para cada aplicación. También un sistema utilizado para el almacenamiento, ya que replica los datos en tiempo infinito. Los datos se almacenan en orden, son duraderos y pueden leerse de forma ya preestablecida ((Dobbelaere and Esmaili, 2017)).

Para esta selección, se ha realizado un estudio de comparación con las herramientas RabbitMQ y Apache Pulsar, analizado cuál se adapta mejor por sus características a nuestro proyecto, obteniendo las siguientes conclusiones:

- RabbitMQ se caracteriza por ser eficiente en términos de cantidad de datos que se escriben, es decir, es más versátil cuando se desea configurar la entrega de mensajes específicos a determinados consumidores ya que emplea vinculación de mensajes con claves, lo que supone una razón para ser descartada ya que en nuestra arquitectura tenemos, a lo sumo, 2 consumidores; la base de datos relacional para la parte de análisis en tiempo real y el *data lake* para la parte *Batch* ((Rabiee, 2018)).
- Uno de puntos fuertes de Apache Kafka para este proceso son sus características para el procesamiento de eventos y la creación de canales de comunicación en tiempo real. Y aunque Apache Pulsar destaca en esta categoría, carece de las sólidas garantías necesarias para las cadenas de transmisión de eventos como ocurre en este proyecto ((Confluent, 2014-2021)).

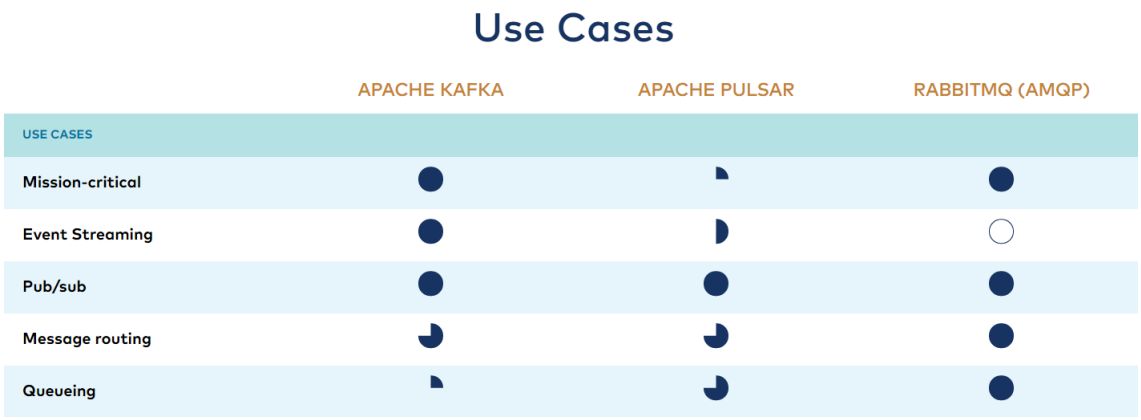


Figura 3.1: Rendimiento de cada herramienta para cada caso de uso.

3.1.4. Almacenamiento

Como resultado a la gran demanda de Big Data y la gran variedad de tipos de datos que existen se han creado muchos modelos, marcos y nuevas tecnologías para proporcionar más capacidad de almacenamiento, procesamiento paralelo y análisis en tiempo real de diferentes fuentes heterogéneas. ((Oussous, Benjelloun, Lahcen, and Belfkih, 2018))

Seleccionar el tipo de software que mejor se adapte al proyecto. Para ello, es necesario tener un diseño claro de la estructura que se quiere implementar para para poder seleccionar el almacenamiento que mejor se adapte a nuestras necesidades ya que puede variar en gran medida dependiendo del tipo de información y el volumen de esta. Es importante la selección de aplicaciones adecuadas para para poder seleccionar las herramientas correctas según las necesidades de cada arquitectura consiguiendo el mejor resultado tanto en tiempo como en eficiencia. ((Pedamkar)).

Sistemas de archivos distribuidos Sistemas de archivos como el sistema de archivos Hadoop (HDFS) ofrecen la capacidad de almacenar grandes cantidades de datos no estructurados de forma fiable en hardware básico. Aunque hay sistemas de archivos con mejor rendimiento, HDFS ha sido diseñado para archivos de datos de gran tamaño y es muy adecuado para la ingesta rápida de datos y el procesamiento masivo.

Bases de datos NoSQL Probablemente estos sistemas de gestión sean la familia más importante de tecnologías de almacenamiento del Big Data. Las bases de datos NoSQL utilizan modelos de datos ajenos al mundo relacional que no necesariamente se adhieren a las propiedades transaccionales de atomicidad, consistencia, aislamiento y durabilidad (ACID).

Algunas de las bases de datos no relacionales más demandadas en el Big Data son las siguientes:

- Cassandra: Es un sistema de almacenamiento distribuido para la gestión de datos estructurados que está diseñado para escalar a un tamaño muy grande a través de muchos servidores de productos básicos basándose sobre una infraestructura de cientos de nodos (posiblemente repartidos en diferentes centros de datos) gestionando el estado persistente de cualquier fallo impulsando la fiabilidad y la escalabilidad

de los sistemas de software que dependen de este servicio. Cassandra destaca por su escalabilidad, alto rendimiento, alta disponibilidad y aplicabilidad asemejándose a una base de datos y compartiendo muchas estrategias de diseño e implementación con las bases de datos. Por otro lado, no admite un modelo de datos relacional completo; en su lugar, proporciona a los clientes un modelo de datos sencillo que admite el control dinámico de la disposición y el formato de los datos. ((Lakshman and Malik, 2010))

- Redis: Es un tipo de servidor apto como memoria rápida para datos. Como sistema de gestión de base de datos (SGBD), Redis ofrece por un lado una base de datos en memoria, guardando todos los datos en la memoria principal, lo que garantiza una respuesta rápida cuando se trabaja con grandes conjuntos de datos sin estructurar y por otro, una base de datos de clave-valor donde para cada entrada se crea una clave (key) mediante la que se puede volver a solicitar la información en cuestión.

  En un servidor Redis, por lo tanto, los datos no se guardan en el disco duro, sino en la memoria principal permitiendo que esta base de datos funcione como memoria caché y también como unidad de memoria principal. ((S.L.U, 2021))

- MongoDB: Es una base de datos que está basada en conjuntos de colecciones. Una colección no tiene un esquema predefinido como las tablas sino que almacena los datos como documentos BSON (objetos con codificación binaria tipo *JSON*). Estos documentos son un conjunto de campos y puede considerarse como una fila en una colección. Puede contener estructuras complejas como listas o incluso un documento completo. Cada documento tiene un campo ID que se utiliza como clave primaria y cada colección puede contener cualquier tipo de documento, pero las consultas y los índices sólo pueden aplicarse a las colecciones. ((Győrödi, Győrödi, Pecherle, and Olah, 2015))

Bases de datos NewSQL Una forma moderna de bases de datos relacionales que pretenden escalabilidad comparable a la de las bases de datos NoSQL, manteniendo las garantías garantías transaccionales de los sistemas de bases de datos tradicionales.

Plataformas de consulta de Big Data Tecnologías que proporcionan interfaces de consulta en frente al resto de tipos de almacenamiento, como los sistemas de archivos distribuidos o las bases de datos NoSQL. La principal preocupación es proporcionar que dicha interfaz sea de alto nivel mediante lenguajes de tipo SQL y lograr una baja latencia en las consultas.

En este proyecto se ha trabajado con bases de datos NoSQL con la finalidad de almacenar toda la información que se genera por parte de la empresa de la manera más rápida y sencilla posible sin necesidad de sufrir ninguna modificación. Se ha llevado a cabo una serie de prototipos con las distintas bases de datos NoSQL más demandadas, con el fin de seleccionar la que mejor se adapta a nuestras necesidades. Hemos creado varios programas en Python y Java estableciendo las conexiones a las bases de datos MongoDB, Cassandra y MySQL, realizando pruebas de inserción con la estructura de los datos que se iban a almacenar siendo finalmente MongoDB la seleccionada.

En nuestro caso, se debe principalmente por cuestiones de uso, MongoDB destaca en la ejecución de consultas de manera dinámica y en la escritura de datos continuada, como

ocurre en nuestro caso, con datos procedentes de Apache Kafka. Además, es el formato *JSON* con el que recibimos la información por parte de la empresa.

Descartamos en primer lugar a Cassandra, ya que está optimizada para almacenar e interactuar con mayores cantidades de datos que los nuestros y para combinar información de diferentes áreas pero en nuestro caso siempre iba a ser tratada información del mismo área.

Por otro lado, MongoDB proporciona tiempos de ejecución más bajo en las operaciones básicas frente a MySQL, lo que la convierte en más idónea para proyectos de gran intensidad de datos como el nuestro, destacando nuestro intereses principalmente por la operación de inserción ya que en este repositorio es la que más se va a ejecutar.

3.1.5. Procesamiento

Cuando hablamos de procesamiento de datos, nos referimos a la etapa más crucial en una arquitectura Big Data. De ello depende la eficiencia al realizar los distintos procesos que nos aportan valor a nuestros datos. El procesamiento lo podemos asemejar a una analogía que aprendimos en un taller de "Arquitecturas Lambda sobre Azure" ((Carlos Espilez y Rubén Villa, 2021)) al que acudimos los integrantes de este proyecto. Esta se puede explicar de una manera clara ya que una de las personas que dictaba el taller mencionaban que el procesamiento de los datos en estas arquitecturas es muy similar al caso de un minero extrayendo oro, en el primer momento en el que este se extrae, posiblemente el metal crudo no tiene el valor que debería, a diferencia de cuando este pasa por los procesos necesarios y después de pulir el metal, este asume el valor que amerita. Con los datos pasa algo parecido, a simple vista podemos obtenerlos de todas partes y de distintas maneras, pero estos no representan ninguna variante para ser analizados, ni aportan información de calidad. Los datos deben representar una realidad, que no sean contradictorios ni estén duplicados, que contengan información completa y que todos los datos estén de acuerdo a su contexto ((Paredes, 2020)).

En los casos que se extraen los datos sin estructurar, aunque existen repositorios como los *data lakes* que nos permiten introducir información sin estructurar, la mayoría de las veces estos deben pasar por un proceso de estructuración en el cual luego se pueden almacenar en repositorios para ser tratados. La idea principal para trabajar de manera eficiente es realizar las inserciones de los datos en repositorios "NoSQL" en los que podemos insertar ficheros en formatos ligeros para su traspaso con grandes cantidades de datos.

Existen herramientas como Apache Spark que nos permiten hacer lecturas de estos formatos de ficheros y crear tablas relaciones de manera local y temporal para tratar los datos de una manera más cómoda.

Para llevar a cabo el procesamiento de los datos en nuestro proyecto, hemos decidido investigar varias alternativas y estudiar que nos ofrece cada una de ellas en conjunto de sus limitaciones. Esta investigación la hemos realizado con tecnologías que nos permitan procesar datos que obtenemos en tiempo real por un lado y con grandes cantidades de datos almacenadas en las las bases de datos utilizadas por otro. En nuestra implementación realizamos análisis de datos que recibimos a medida que se reciben eventos en tiempo real como el alta de nuevos clientes en la empresa o cada una de sus transacciones realizadas. También requerimos del análisis en conjunto de de estos, requiriendo de tiempo y eficiencia

para procesar los análisis deseados y generar informes para la toma de decisiones para la creación de nuevas promociones por parte de la empresa.

Entre las variantes de las herramientas de procesamiento investigadas, podemos nombrar las siguientes:

Apache Storm Es un sistema que procesa datos de manera distribuida en tiempo real. Este sistema facilita el procesamiento de flujos de datos ilimitados. Entre sus usos principales podemos recalcar el análisis en tiempo real, la realización de aprendizaje automático en línea, computación continua, y ETL (Extract, Transform and Load), ETL es un proceso que permite extraer datos de distintas fuentes, proporcionarles formato y luego almacenarlos en un sistema de almacenamiento.

Esta herramienta se encarga de la paralelización, la partición y los reintentos en caso de fallas cuando sea necesario de manera automática. Usa flujos de tuplas, estas representan listas de valores con un nombre que las referencia, estas listas pueden contener objetos de cualquier tipo. El funcionamiento se estructura en tres conceptos básicos: *spouts, bolts, and topologies* ((Foundation, 2019a))

Un *Spoult*, representa la fuente de flujos. Lee de colas de mensajería como Kafka o puede generar su propio *stream*, leyendo de algún sitio como la API de *streaming* de Twitter que comentamos anteriormente.

Los *Bolts* son capaces de procesar cualquier cantidad de flujos de entrada y generar cualquier número de nuevos flujos de salida. La mayor parte de la lógica de computación la tenemos en los *Bolts*, como las funciones, los filtros para consultas, las agregaciones de los flujos de datos y las conexiones a los repositorios de almacenamiento.

Topology, es una red de *Spoults* y *Bolts*, las aristas que unen los nodos de esta red representa un *bolt* que se suscribe al flujo de salida de algún otro *spoult* y *bolt*. Cuando se despliega una topología esta se ejecuta de manera indefinida.

Como ventajas principales de esta herramienta podemos mencionar las siguientes:

- Es de código abierto, gratuito y se puede utilizar con cualquier lenguaje de programación.
- Es un sistema escalable, Las topologías son paralelas y se ejecutan en *clusters* de máquinas. Las partes de la topología pueden escalarse individualmente y en tiempo de ejecución.
- Permite integrarse con bases de datos y colas de mensajería mayormente utilizadas como: RabbitMQ/AMQP y Apache Kafka. ((Foundation, 2019b))
- Es tolerante a fallos, cuando los procesos mueren, Storm se encarga de reiniciar los procesos automáticamente en un nodo que esté disponible.
- Ofrece garantías de procesamiento, procesando en su totalidad todas tuplas de flujos. Los mensajes sólo se repiten cuando se produce un fallo.
- Esta herramienta nos ofrece un modo local de ejecución que simula el funcionamiento de un *cluster* de Apache Storm. Esto viene bien para la realización de pruebas en nuestros ordenadores.

- El paralelismo puede procesar una gran cantidad de mensajes con una latencia muy baja.
- "Apache Storm was benchmarked at processing one million 100 byte messages per second per node on hardware with the following specs: Processor: 2x Intel E5645@2.4Ghz , Memory: 24 GB" ((Foundation, 2019c))

Apache Flink Es un *framework* (Marco de trabajo) y un motor de procesamiento distribuido para cálculos de flujos de datos tanto en tiempo real como por conjuntos grandes de datos (Batch). Posibilita su ejecución en administradores de *clusters* comúnmente utilizados.

Flink mantiene el estado de los procesos en la memoria siempre y cuando no sobrepase los recursos disponibles, en los casos de agotar los recursos almacena los datos en disco. Para la tolerancia a fallas, realiza comprobaciones asíncronas del estado local en un almacenamiento duradero. ((Foundation, 2014-2021)).

Para el procesamiento de los datos en tiempo real, Flink exige que la ingesta de los datos tenga un orden determinado en el que van ocurriendo los eventos, para mantener la integridad de los resultados, caso que no aplica para el procesamiento por lotes de datos.

Flink procesa los datos a medida que se van recibiendo los eventos, esto mejora notablemente la latencia en ejecución y un mejor rendimiento. La gestión de la memoria al procesar los datos lo realiza de manera automática.

Apache Hadoop Apache Hadoop es un *framework* de código abierto para ejecutar aplicaciones en grandes *clusters*. La herramienta usa el paradigma computacional Map/Reduce que se basa en dividir la aplicación en muchos fragmentos pequeños de trabajo donde cada uno puede ejecutarse en distintos nodos de un *clusters*.

Esta herramienta proporciona un sistema de archivos distribuido (HDFS) que almacena los datos en los nodos de cálculo permitiendo distribuir las cargas de trabajo y realizarlas en forma paralela. También permite realizar procesamiento *Batch* distribuido en *clusters* de manera simple.

Proporciona alta escalabilidad desde servidores individuales hasta miles de máquinas, cada una de las cuales ofrece computación y almacenamiento local. Está diseñada para gestionar los fallos de manera automática, ofreciendo así un servicio de alta disponibilidad sobre un *clusters* de ordenadores.((Foundation))

A continuación mencionamos unas de las tantas ventajas que nos ofrece la herramienta:

- Ofrece un sistema de archivos distribuido que proporciona acceso de alto rendimiento a los datos de la aplicación.
- Ofrece mecanismos de gestión de *clusters* para ejecutar aplicaciones.
- Permite realizar procesamiento distribuido de los datos.
- Se puede integrar con Apache Spark.

Apache Spark Es un motor de análisis unificado para el procesamiento de datos a gran escala ((Foundation, 2018a)). Entre las funciones más destacadas, permite realizar análisis de datos en tiempo real, aprendizaje automático y ETL.

Su funcionamiento se basa en distribuir aplicaciones que requieren de la realización de múltiples tareas de manera simultánea.

Las ventajas más destacables al usar este sistema son las siguientes:

- Proporciona un alto rendimiento tanto para procesar los datos en memoria bien sea por lotes o en tiempo real. Este utiliza un optimizador de consultas SQL y un motor de ejecución física.
- Es muy sencillo de programar y soporta tecnologías de programación comunes como Java, Scala, Python y SQL.
- Ofrece acceso a diversas fuentes de datos distribuidas, repositorios en la nube o colas de mensajería como Kafka.
- Permite la ejecución de manera local sin necesidad de levantar un *cluster*. Este se encarga de simularlo.

Entre las distintas librerías que nos ofrece Apache Spark, es importante explicar un poco más a fondo dos de ellas que son las que hemos utilizado mayormente en nuestra implementación.

Spark Streaming Esta nos permite desarrollar programas de procesamiento en *Batch* y *Real Time* de la misma manera. Este hecho lo demostraremos en el capítulo de la implementación de nuestro proyecto. Nos ofrece tolerancia a fallos, en estos casos Apache Spark se encarga de recuperar el trabajo perdido de forma inmediata. ((Foundation, 2018b)).

Spark SQL Nos permite realizar las consultas a los repositorios en los que tenemos almacenados nuestros datos estructurados. El acceso a los datos es unificado, en la implementación podemos demostrar que la lectura de los datos de un fichero local se realiza de la misma manera que para los que están en un repositorio en la nube. La conexión a las fuentes de datos se puede realizar de manera estándar a través de los drivers de JDBC y ODBC. Spark SQL incluye un optimizador basado en costes, almacenamiento en columnas y generación de código para que las consultas sean rápidas. Al mismo tiempo, escala a miles de nodos y a consultas de varias horas de duración utilizando el motor Spark, que proporciona una tolerancia total a los fallos en mitad de la consulta ((Foundation, 2018c)).

¿Por qué escogimos Apache Spark? Debemos mencionar que la tarea de seleccionar una de las tecnologías a utilizar ha sido una actividad compleja, ya que tienen ciertas similitudes a niveles de compatibilidad y tiempos de procesamiento.

Para llevar a cabo esta tarea hemos estudiado las comunidades que ofrecen soporte para cada una de ellas, los resultados han sido favorables para el caso de Apache Spark. Esto es un dato muy importante ya que para implementar la arquitectura necesitaremos información que nos respalde para llevar a cabo nuestros desarrollos.

Otra característica importante, es que Spark nos permite combinar operaciones de consulta SQL, con procesamiento en tiempo real y realización de análisis complejos en un mismo programa debido a las librerías que este dispone. Esto es importante ya que en nuestro caso tendremos los datos de manera estructurada en los repositorios de almacenamiento.

Spark también nos permite integrarnos de manera muy sencilla a las colas de mensajería Kafka y a los repositorios en la nube en los que se alojan los datos a utilizar en nuestra implementación. De ellos hablaremos en la sección de Cloud Computing.

Gracias a que Spark nos ofrece una documentación completa para lenguajes como Python y Scala, hemos decidido optar por utilizar ambos lenguajes en nuestra implementación, utilizando Python para la capa de procesamiento en tiempo real y Scala para el caso del procesamiento de datos por lotes.

Comparación de Spark con Hadoop Spark se encarga de que el análisis de datos sea rápido de escribir y de ejecutar. A diferencia de otros sistemas que también utilizan el paradigma MapReduce como Hadoop. Spark permite la consulta de datos distribuidos en memoria en lugar de utilizar la escritura y lectura disco ya que estas operaciones son poco eficiente el procesamiento. Spark no soporta un sistema de archivos distribuido por sí mismo, depende totalmente de Hadoop, si se requiere de un HDFS.

Hadoop es un sistema de procesamiento por lotes, trabaja con un gran conjunto de datos estáticos. Spark viene siendo una evolución de Hadoop y no un rival directo.((SCHOOL, 2018))

Comparación de Spark con Storm y Flink Storm intenta simular el mismo principio que utiliza Hadoop pero en tiempo real, al procesar los datos a medida que se van generando, obtiene mejores tiempos de ejecución y menos latencia que Spark. Storm se puede complementar con Hadoop ya que no es eficiente al procesar grandes cantidades de datos. ((da Silva Morais, 2015))

En el caso de Flink, al igual que Storm, nos ofrece mejor rendimiento en el procesamiento de los datos en tiempo real que Spark, debido a que este procesa los datos a medida que se van generando, sino que ejecuta una serie de micro batches por ventanas de tiempo asignada, con el tiempo asignado este acumula una serie de datos para luego procesarlos. Esta razón hace que Spark sea menos eficiente que Flink y Storm.

En el caso de Storm por su complejidad de implementación y su carencia del procesamiento *Batch* que es indispensable en una de las capas de nuestra arquitectura, nos hemos visto obligados a descartar esta alternativa.

Flink parecía ser la opción ideal, pero es un proyecto mucho más nuevo, careciendo de una comunidad tan grande como la de Spark. También tiene algunos problemas de escalabilidad a nivel de procesamiento *Batch*. Esto no nos afectaría para nuestra implementación, pero como es un proyecto para una empresa que puede seguir evolucionando, nos preocupamos por que sean altamente escalables.

Otra razón importante de la elección de Spark es que nos permite reutilizar el código para el procesamiento por conjuntos de datos y en tiempo real con un nivel eficiente de procesamiento.

3.1.6. Visualización

Una vez realizado los procesos de extracción, almacenaje y procesamiento de los datos llega el momento de mostrar esos datos de una manera clara y comprensible. La visualización de datos es la presentación de los datos en un formato pictórico o gráfico, proporciona a los usuarios medios intuitivos para explorar y analizar interactivamente los datos, permitiéndoles identificar eficazmente patrones interesantes o inferir correlaciones y causalidades. ((Keim, Qu, and Ma, 2013))

Hay una gran variedad de herramientas que nos permiten una visualización de los datos pero todas ellas se caracterizan por la búsqueda de la menor latencia posible mediante el uso de datos precalculados o paralelizando el procesamiento de estos y su interpretación. Se debe tener en cuenta que estas herramientas tienen que ser capaces de tratar con datos semi estructurados y no estructurados porque son los datos que nos encontramos en el Big Data.

Tableau Es una herramienta de visualización de datos interactiva que se centra en el Business Intelligence ofreciendo una amplia gama de opciones de visualización personalizadas. Es rápido y flexible y soporta la mayoría de los formatos de datos y la conexión a varios servidores. Por contra los servicios gratuitos no ofrecen una gran variedad.((Ali, Gupta, Nayak, and Lenka, 2016))

Microsoft Power BI Power Bi de Microsoft es una herramienta gratuita que se caracteriza por ser tablas dinámicas de Excel llevadas a un nivel mayor de procesamiento con la elaboración de diagramas, gráficos y tablas de una manera sencilla. Por contra, Power BI tiene algunas limitaciones en la creación de fórmulas y en la configuración de las interfaces personalizadas.((AbsentData, 2020))

Infogram Es una plataforma de visualización de datos en línea, tiene un editor fácil gracias a las plantillas ya preparadas y a las sencillas opciones de personalización incluidas en la opción gratuita. Por otro lado, es difícil presentar múltiples campos en Infogram, lo que dificulta la comparación de varias características de los datos entre sí para una visualización y un análisis más complejos.

Plotly Es un potente software de código abierto que permite a los usuarios hacer gráficos interactivos, ya sea de manera sencilla desde la interfaz en línea o por el contrario se puede ajustar manualmente y utilizar como una biblioteca de programación para los lenguajes de codificación Python, R y JavaScript. ((Science, Oct 17, 2018))

Grafana Es una herramienta diseñada especialmente para hacer análisis de series temporales fundamentalmente en tiempo real. Se caracteriza por ser una herramienta gratuita destacando por sus diferentes tipos de autenticación y niveles de seguridad junto con su sistema de alertas y notificaciones. Por contra, presenta algunas limitaciones en la organización y diseño de los dashboards y no es auto gestionado.((Keepler))

Después de este análisis, la seleccionada para nuestro proyecto fué Grafana ya que aunque no es una herramienta tan potente como Power BI o Tableau entre otras, el hecho de que esté enfocada a la visualización de series temporales ha sido lo que hizo decantarnos por ella debido a que en nuestro proyecto se va a mostrar la información al mismo

tiempo que se va recibiendo, pudiéndose mostrar en rangos de tiempo seleccionados por el usuario.

3.2. Cloud Computing

El almacenamiento de los datos supone una parte crucial para cualquier proceso de análisis que se desee realizar con ellos. Al estar hablando de cantidades muy grandes de información se convierte en un problema, teniendo en cuenta también, que el acceso a esta información tiene que ser lo más rápido y cómodo posible para su manipulación posterior. Es aquí donde aparecen los entornos Cloud, proveedores de servicios externos que almacenan dicha información y programas de cómputo para el análisis de estos. Son grupos de miles y miles de ordenadores conectados por la red recibiendo el nombre de granja de servidores y con un diseño como el que se observa en la fotografía 3.2. Cada grupo es un centro diseñado para proporcionar a los usuarios almacenamiento y aplicaciones de procesamiento de los datos allí situados. Los usuarios pueden ejecutar interfaces de aplicaciones, como procesadores de texto y búsquedas en la web, a través de navegadores web. Los datos son accesibles desde las bases de datos con un fácil acceso tanto para la inserción como para la extracción ((Li, 2013)).

Figura 3.2: Granja de servidores.

3.2.1. Fundamentos y tipos

El Cloud Computing está transformando la forma en que las empresas y organizaciones adquieren y gestionan sus recursos informáticos. Este servicio se basa en proporcionar un modelo de TI en el que el proveedor de la nube se hace responsable de una serie de actividades informáticas imprescindibles para su actividad, como la instalación de hardware y software, actualizaciones, mantenimiento, copias de seguridad, almacenamiento de datos o seguridad. Con ello, se consigue reducir gastos de capital de TI y los costes de funcionamiento mediante la compra de recursos tecnológicos a la carta dependiendo de las necesidades. Estos servicios también incluyen entornos para el desarrollo de aplicaciones y el acceso a tecnologías clave, software y personal informático cualificado que, de otro

modo, podrían ser demasiado costosos y difícil de obtener y mantener. ((Garrison, Kim, and Wakefield, 2012))

La computación en nube se organiza en tres modelos de servicio estándar ((Fehling, Leymann, Retter, Schupeck, and Arbitter, 2014)):

- Infraestructura como servicio (IaaS): El proveedor ofrece procesamiento, almacenamiento y conectividad de red, normalmente en forma de servidores o servidores virtualizados. Utilizando estos recursos de TI, los clientes pueden alojar sus propios sistemas operativos y aplicaciones.
- Plataforma como servicio (PaaS): El proveedor ofrece un entorno de ejecución de las aplicaciones. Los servidores, el almacenamiento, la conectividad de red y otros recursos se ocultan al cliente, que crea una aplicación compatible con el proveedor. Los proveedores pueden admitir lenguajes de programación personalizados y ofrecen bibliotecas para acceder a la funcionalidad del entorno.
- Software como servicio (SaaS): El proveedor ofrece una aplicación completa. La infraestructura en la nube que se emplea está oculta para el cliente pero acceden a esta aplicación a través de interfaces de usuario pudiendo configurar esta aplicación.

3.2.2. Problemas e Inconvenientes

Uno de los principales retos de trasladar las aplicaciones a la nube es, principalmente, la necesidad de dominar varios lenguajes de programación y entornos operativos. En muchas aplicaciones en la nube, un proceso back-end depende de una base de datos relacional, por lo que parte del código se escribe en SQL u otro lenguaje de consulta. En el lado del cliente, es probable que la lógica del programa se implemente en JavaScript incrustado en documentos HTML. Entre la base de datos y el cliente hay una aplicación de servidor que puede estar escrita en un lenguaje de scripting (como PHP, Java y Python). La información que se intercambia entre las distintas capas se codifica probablemente en alguna variante de XML. ((Hayes, 2008))

Otro de los inconvenientes principales a tener en cuenta es la seguridad, siendo esta una responsabilidad que se divide entre muchas partes potenciales, incluyendo tanto al usuario como al proveedor de los servicios y a cualquier proveedor de terceros en el que los usuarios confíen para el software o las configuraciones sensibles a la seguridad.

El usuario de la nube es responsable de la seguridad a nivel de aplicación mientras que el proveedor de la nube es responsable de la seguridad física y de hacer cumplir las políticas de los cortafuegos externos. La seguridad de las capas intermedias de la pila de software se comparte entre el usuario y el operador; cuanto más bajo es el nivel de abstracción expuesto al usuario, más responsabilidad conlleva.

Aunque la computación en nube puede facilitar la seguridad de cara al exterior, plantea el nuevo problema de la seguridad de cara al interior. Los proveedores de la nube deben protegerse contra el robo o los ataques de denegación de servicio por parte de los usuarios al igual que los usuarios deben estar protegidos unos de otros.

Dentro de la seguridad, existe también la preocupación en materia de protección del usuario frente al proveedor. El proveedor controla la información y software del cliente, lo que elude la mayoría de las técnicas de seguridad conocidas, teniendo el control y

acceso absoluto de la información.((Armbrust, Fox, Griffith, Joseph, Katz, Konwinski, Lee, Patterson, Rabkin, Stoica, et al., 2010))

El principal mecanismo de seguridad del Cloud Computing es la virtualización que protege contra la mayoría de los intentos de ataque entre usuarios o a la propia infraestructura de la nube. Este mecanismo convierte la versión virtual de un recurso o de un dispositivo en un servidor, un sistema operativo, una red o un dispositivo de almacenamiento haciéndolos mas flexibles y adaptables en la gestión de su seguridad.((Kumar and Charu, 2015))

3.2.3. Futuro del Cloud Computing

Es muy grande la importancia que ha adquirido y está adquiriendo el Cloud Computing en las empresas y administraciones con el paso del tiempo. Es por ello que estos servicios presentan numerosos retos y desafíos para la mejora de sus servicios.

Uno de los más importantes, como ya hemos indicado anteriormente es el de la seguridad. Cualquier mejora en la reducción de pérdida de datos, en un menor control sobre los procesos del cliente, en la reducción de cualquier ataque interno, en la falta de aspectos legales o en la falta de migración de un proveedor a otro seguirá haciendo del Cloud Computing una herramienta indispensable para cualquier corporación. Es por eso, que la virtualización cobra un lugar muy importante en este reto ((Birje, Challagidad, Goudar, and Tapale, 2017)).

Otro de los desafíos que presenta el Cloud Computing es la mejora de su arquitectura. La mayor parte de la infraestructura existente de los servicios en la nube está compuesta por recursos de computación y almacenamiento ubicados en centros de datos de un único proveedor. Esto ofrece facilidades y ventajas evidentes, sin embargo, consume mucha energía para mantenerlo operativo y como cualquier otro modelo informático centralizado, son susceptibles de sufrir fallos en un solo punto paralizando el resto de servicios. Además, cabe destacar que los centros de datos pueden estar geográficamente alejados de sus usuarios lo que puede suponer que las aplicaciones que utilizan o generan datos sensibles o personales pueden tener que ser almacenadas en un país diferente al de su origen, además del tiempo de envío de la información.

En los últimos años se han propuesto modelos alternativos de uso de la infraestructura de la nube para sustituir los centros de datos de un único proveedor surgiendo estrategias para aprovechar los recursos de múltiples proveedores. Este modelo, llamado nube híbrida, satisface demandas de ráfagas de información con un gran beneficio para el manejo de datos sensibles. Sin embargo, se tiene que tener en cuenta el ancho de banda, la latencia y las topologías de red para acceder a una nube pública desde una nube privada. Las limitaciones de la red pueden resultar en una nube híbrida ineficaz ((Varghese and Buyya, 2018)). Es por eso que las nubes híbridas suponen un reto importante a mejorar en durante los próximos años.

Debido al auge, la importancia y la demanda que tienen estos servicios junto con las grandes posibilidades de mejora que se pueden conseguir, es de esperar que el futuro del Cloud Computing estará marcado por un crecimiento de proveedores de estos servicios, un aumento y mejora de los servicios ofrecidos, una mejor legalidad y unas mejores prácticas de uso.

Capítulo 4

IMPLEMENTACIÓN

Como subestructura presentamos nuestra arquitectura Big Data para resolver cada uno de los objetivos planteados al inicio del trabajo, en este capítulo comentaremos cada uno de los detalles de implementación para llevar a cabo nuestro ecosistema, se explican los requisitos previos de configuración, el funcionamiento y los resultados obtenidos con nuestra solución.

4.1. Diseño y Estructura

El diseño de nuestra arquitectura está basado en la construcción de un ecosistema que realice el tratamiento de grandes cantidades de información para la realización de análisis generales. En este caso lo aplicaremos para tratar información de clientes, promociones y ventas de la empresa. Por un lado queremos visualizar y controlar todas las posibles ventas y altas de promociones de los clientes en las distintas tiendas de España en tiempo real y por otro, realizar un análisis detallado sobre esta información para una futura toma de decisiones en la creación de nuevas promociones particulares aplicables a clientes.

La estructura de la arquitectura la dividimos en cinco partes fundamentales:

- Extracción de la información mediante consultas al *API* de las fuentes internas proporcionadas por la empresa. En nuestro caso y para este proyecto este apartado se basa en replicación en generadores de estos datos con la misma estructura pero con datos no sensibles de clientes, ni de productos seriados reales. Una vez generada dicha información se inserta en las colas de mensajería. Se puede observar en el recuadro azul de la figura 4.1.
- Almacenamiento de la información de las colas de mensajería sin ningún tipo de modificación en MongoDB, haciendo de este nuestro repositorio con el histórico de datos de la empresa para un posible análisis futuro si fuese necesario. Se puede observar en el recuadro negro en la figura 4.1.
- Tratamiento de los datos al mismo tiempo que se obtienen de los eventos producidos por los generadores de datos. En este apartado implementamos la lógica correspondiente para procesar dicha información según se va recibiendo y su posterior almacenamiento en la base de datos relacional de Azure SQL Server alojada en los servidores Cloud. Se puede observar en el recuadro naranja inferior *Speed Layer* en la figura 4.1.

- Guardado de la información sin procesar mediante lotes grandes de datos en Azure Storage para posteriormente realizar un proceso de análisis de esta información en periodos de tiempo más largos y con la finalidad de obtener unos resultados más concretos y detallados. Este apartado se corresponde con el recuadro verde superior *Batch Layer* de la figura 4.1.
- Visualización de la información. Por un lado, mediante entornos gráficos que nos permiten la visualización de manera inmediata de los resultados de los procesos de análisis y por otro se producirán una serie de informes que se generan en intervalos de tiempo determinados y con una información más completa. A esta parte del proyecto hace referencia el recuadro rojo de la figura 4.1.

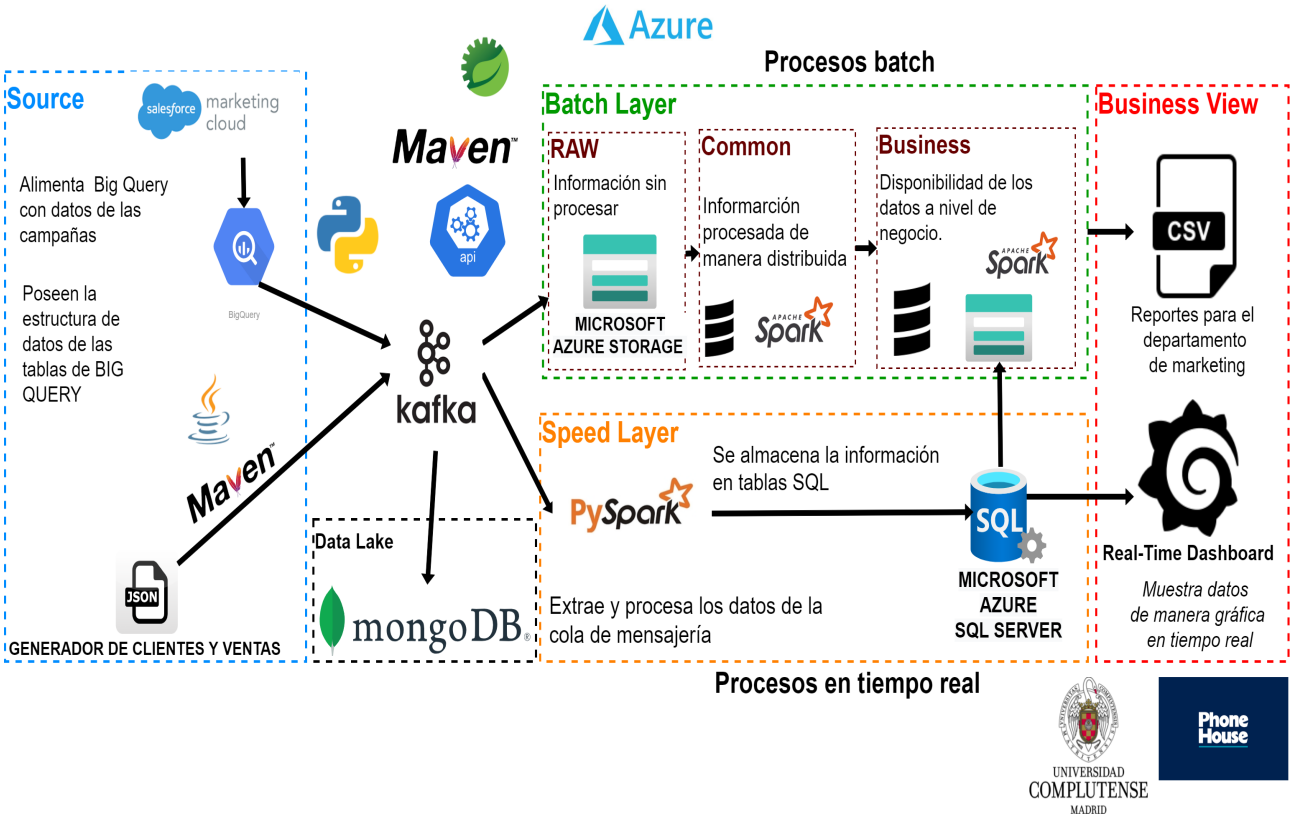


Figura 4.1: Gráfica de la arquitectura Lambda implementada.

Este diseño hace referencia a una Arquitectura Lambda en un sistema Big Data con la idea de crear los sistemas de tratamiento de la información combinando el modo *Batch* y el tiempo real *stream* de manera independiente, obteniendo un mejor rendimiento y una menor latencia a la hora de realizar las distintas tareas.

La ventaja principal de implementar esta arquitectura en nuestro proyecto es que nos permite asegurar un alcance completo de todos los datos a lo largo del tiempo de manera confiable y a su vez procesar datos en tiempo real, mejorando con esto la toma de decisiones por parte de negocio. En este caso, es el departamento de marketing de la empresa los encargados de visualizar y analizar estos datos.

Podemos mencionar algunas de las ventajas adicionales que nos proporciona esta arquitectura ((Hasani, Kon-Popovska, and Velinov, 2014)):

- Modelo escalable, este nos permite un crecimiento a futuro de la arquitectura sin tener que realizar demasiadas modificaciones. De igual manera en este proyecto se aplica a una cantidad inferior de datos que sin mucha complejidad podría ser ampliable a niveles empresariales solo aumentando la cantidad de recursos.
- Modelo de software sostenible que no requiere de una administración técnica o mantenimiento.
- La disponibilidad de los datos es confiable y ofrece garantías de respuesta a las posibles peticiones que se realicen, ofreciendo respuestas en todo los casos.

Como principal inconveniente cabe mencionar la complejidad en su desarrollo, ya que los códigos para cada capa se deben realizar en paralelo y estos son totalmente independientes pese a que apunten a los mismos repositorios de almacenamiento, dificultando su depuración de posibles errores en el código.

4.2. Software implementado

Para las diferentes partes de esta arquitectura se han empleado gran variedad de aplicaciones, herramientas y tecnologías buscando siempre la mayor eficiencia y simplicidad para nuestro proyecto. Muchas estas tecnologías eran desconocidas por nuestra parte pero después de las investigaciones y la creación de prototipos hemos sabido seleccionar e implementar las que mejor se adaptan a este proyecto.

4.2.1. Extracción de datos

Pruebas de concepto

Siendo uno de los objetivos de este proyecto la extracción de datos mediante una API hemos trabajado en construir un prototipo funcional para probar este tipo de extracción. Desarrollamos un programa en lenguaje Python donde nos centramos en consultar la API de la red social Twitter. Como primer requisito se necesita registrarse en esta red social e ingresar una serie de datos personales que solicita la aplicación.

Una vez registrados, se debe dar de alta como cliente del API, para que Twitter proporcione las credenciales de acceso. La documentación de esta API la podemos encontrar en la *Web* ((TWITTER, 2021)) sin la obligación de estar registrado. En ella podemos encontrar la información de los servicios que se pueden solicitar y la estructura que debe contener la petición a realizar. Es importante mencionar que las compañías que ofrecen estos servicios de extracción tienen un límite de datos a consultar basados en su política y asignación de recursos. Esto se debe a que cada vez que realizamos peticiones, los servicios internos de dichas empresas son los que se encargan de consultar los datos utilizando sus propios recursos para enviarlos a los usuario, con estos límites evitan la sobrecarga de sus sistemas.

En nuestra implementación incluimos una librería de Python llamada *Tweepy* ((Revision, 2009,2020)) que permite autenticarnos y realizar peticiones de manera muy sencilla. Empezamos por llamar al método *OAuthHandler(consumer_key, consumer _secret)* a la que se le pasa como argumentos las credenciales del cliente y luego una llamada al *set_access_token* en el cual añadimos los *token* proporcionados por la aplicación.

Una vez configurada la variable de autenticación realizamos la llamada al método *API(autenticación)* para autenticarnos. Esta nos proporciona una referencia al servicio y con esta hemos realizado una serie de peticiones al API con varios métodos que nos proporciona la librería, entre ellos:

- *api.me()*: proporciona información de la cuenta del usuario autenticado.
- *api.get_user("user")*: proporciona información de la cuenta de un usuario.
- *api.followers("user")*: proporciona información de la cuenta de los seguidores de un usuario.

Con esto conseguimos unas primeras tomas de contacto con APIs profesionales, utilizando documentación estructurada. Los datos obtenidos se representan formatos tipo *JSON* que es un formato ligero de intercambio de datos ((Lennon, 2009)). En la figura 4.2 se pueden visualizar los resultados obtenidos de una consulta a los datos del usuario Adidas.

Figura 4.2: Resultados obtenidos con el prototipo de extracción de datos de Twitter.

Aunque en la imagen se aprecian pocos datos, los resultados son muy amplios y los datos se pueden filtrar para realizar cualquier petición proporcionando la información necesaria para que se aproximen a los objetivos planteados.

Partiendo de las pruebas anteriores y teniendo una noción de como extraer datos a través de una API se construyó un segundo prototipo fundamental de nuestra implementación. Se basó en el acceso a los datos de las tablas alojadas en "Google Cloud BigQuery" por parte de la empresa The Phone House. Este es un almacén de datos alojado en la nube utilizado a niveles prácticos de la empresa para guardar una serie de datos para luego ser

analizados. Para realizar nuestro prototipo nos dieron acceso a dos tablas, por parte de la empresa, de las numerosas que tienen alojadas en esta plataforma. Estas tablas que nos proporcionaron representan los datos de clientes y de las ventas, suficiente para realizar nuestra arquitectura y cumplir con los objetivos propuestos.

El proceso realizado para autenticarse en el *API* de Google, es similar al anterior, basta con agregar la librería de "google-cloud-bigquery" ((Cloud, 2020)) y seguir el procedimiento con las nuevas credenciales.

En este caso no tuvimos autorización para demostrar los resultados de las consultas ya que entre los datos se obtiene información sensible que no puede ser representada.

Con este prototipo logramos conocer la estructura de los datos internos utilizados por la empresa y empezamos a recopilar información a utilizar en nuestra arquitectura y para los generadores.

Generadores de datos

Partiendo de esto damos paso a la construcción de los generadores de datos, estos representan las fuentes de información con las que partimos en la arquitectura implementada. Se encargan de replicar la información estructurada que se obtiene de las bases de datos proporcionadas por la empresa. Generan datos con el mismo formato y estructura, pero con un sentido de aleatoriedad en los datos más sensibles. Los motivos principales se deben, por un lado al no tener aprobación de parte del departamento legal de la empresa para que nos otorgara autorización para trabajar con datos reales, por otro lado tenemos el Reglamento General de Protección de Datos (RGPD) que no permite el tratamiento de los mismos por una entidad externa a la empresa sin el consentimiento de los clientes.

Se desarrollaron dos proyectos en Java como lenguaje de programación, que contienen la funcionalidad de generar datos de clientes y transacciones de venta en intervalos de tiempo determinados, estos se generan en formato *JSON*.

En la figura 4.3 podemos observar el esquema de los datos proporcionados por la empresa para los detalles de cliente. En ella tenemos el nombre de los campos, el tipo de dato, los datos obligatorios y una descripción del significado de cada uno.

BQ TABLE: customer_detail

Nombre del campo	Tipo de dato	Indicador de posible nulo	Descripción del campo
NAME	**TIPO**	**NULLABLE**	**DESCRIPTION**
CUSTOMER_ID	STRING	No	Identificador de cliente.
MAIL	STRING(250)	Yes	Email de contacto del cliente
CUSTOMER_ID_PH	NUMBER	No	Identificador de cliente para el sistema de facturación interno.
PROMO_BIRTHDAY	1/0	No	Indicador de promo de cumpleaños. 1=activada / 0=desactivada.
BIRTHDAY_PROMO_SINCE	DATE	Yes	Fecha de inicio de la promoción de cumpleaños.
BIRTHDAY_PROMO_TO	DATE	Yes	Fecha de fin de la promoción de cumpleaños.
BIRTHDAY_DISCOUNT_REASON	STRING	No	Razón de descuento cumpleaños.
BIRTHDAY_DISCOUNT_DESCRIPTION	STRING	No	Detalle razon de descuento cumpleaños.
VALID_MAIL	1/0	No	1 si el cliente tiene un email valido / 0 en caso contrario.
DFL_LUZ	1/0	No	1 si el cliente tiene contratado el servicio de energía / 0 en caso contrario.
DFL_GAS	1/0	No	1 si el cliente tiene contratado gas / 0 en caso contrario.
DFL_TELCO	1/0	No	1 si el cliente tiene contratado servicios de telefonía / 0 en caso contrario.
DFL_INSURANCE_HOME	1/0	No	1 si el cliente tiene contratado seguro de hogar / 0 en caso contrario.
PERMISSION_COMUNICATION	1/0	No	1 si el cliente tiene habilitado los permisos para recibir comunicaciones / 0 en caso contrario.

Figura 4.3: Campos de los datos de los clientes.

En la figura 4.4 podemos observar el esquema de los datos de ventas.

BQ TABLE: sales_detail			
Nombre del campo	Tipo de dato	Indicador de posible nulo	Descripción del campo
NAME	**TIPO**	**NULLABLE**	**DESCRIPTION**
CUSTOMER_ID_PH	NUMBER	Yes	Identificador de cliente para el sistema de facturación interno.
DISCOUNT_REASON	STRING(150)	No	Razón de descuento.
DISCOUNT_DESC	STRING(200)	No	Descripción de descuento.
CLOSED_DATE	DATETIME	No	Fecha de cierre de la transacción
FAMILIA_LTV	STRING(100)	Yes	Grupo al que pertenece el producto o servicio vendido.
TRANSACTION	STRING(20)	No	Identificador de transacción.
PRODUCT_QUANTITY	NUMBER	No	Cantidad de productos en la transacción.
TOTAL_DISCOUNT_AMOUNT	FLOAT	No	Importe total de descuento.
TOTAL_RETAIL_AMOUNT	FLOAT	No	Importe total de la venta.

Figura 4.4: Campos de los datos de las ventas.

En la figura 4.5 y 4.6 tenemos una muestra de la salida que proporciona cada uno de los generadores de datos, siguiendo el esquema mencionado anteriormente.


```
{
   "BIRTHDAY_PROMO_TO": "2021-08-08 12:07:23 UTC",
   "DFL_TELCO": 0,
   "DFL_INSURANCE_HOME": 0,
   "CUSTOMER_ID": "PH_0086799",
   "DFL_LUZ": 0,
   "BIRTHDAY_DISCOUNT_REASON": "MICUMPLE",
   "BIRTHDAY_DISCOUNT_DESCRIPTION": "Descuento aplicable a: Telefonos Libres y con Contrato (Altas, Portas y Puntos).",
   "PERMISSION_COMUNICATION": 0,
   "DFL_GAS": 1,
   "MAIL": "fdazmbgbo@gmail.com",
   "PROMO_BIRTHDAY": 0,
   "BIRTHDAY_PROMO_SINCE": "2021-07-24 12:07:23 UTC",
   "CUSTOMER_ID_PH": 86799
}
```


Figura 4.5: JSON creado por el generador de clientes.


```
{
   "DISCOUNT_DESC": "Descuento de empleados",
   "TOTAL_RETAIL_AMOUNT": 90,
   "PRODUCT_QUANTITY": "1",
   "SHOP_ID": 4747011,
   "CUSTOMER_ID": 8923576,
   "DISCOUNT_REASON": "EMPLEADOS",
   "TOTAL_DISCOUNT_AMOUNT": 10,
   "FAMILIA_LTV": "POSTPAY_ORANGE",
   "TRANSACTION": "T101417691",
   "SHOP_CITY": "PALENCIA",
   "CLOSED_DATE": "2020-12-13 01:14:47"
}
```


Figura 4.6: JSON creado por el generador de ventas.

El generador de detalles de ventas consta con una serie de enumerados. Uno en el que se encuentran las ciudades en la que la empresa posee tiendas abiertas y un código que las identifica. Otro enumerado contiene todos los tipos de familia por producto que están definidos en la empresa y el coste por producto. Finalmente tenemos uno que especifica los tipos de descuento disponibles, con su nombre y descripción.

Adicional a esto el generador posee una clase que se encarga de generar datos aleatorios coherentes para cada uno de los campos del esquema definido. La lógica de cada cálculo aleatorio se basa en parámetros que se deben cumplir para que se contenga valores coherentes, como por ejemplo activar una promoción si se realiza en el rango de fecha especificado, entre otros tipos de requisitos que deben cumplir para simular un dato real.

Se tiene una clase de apoyo que recibe toda la información y la formatea para almacenarla en ficheros *JSON*.

Finalmente la clase principal que posee la lógica para generar los datos en intervalos de tiempo determinados mediante pausas de la ejecución. Esta también contiene la conexión al SQL Server de Azure que explicaremos en el apartado de procesamiento en tiempo real. Básicamente esta conexión la realizamos para antes de generar una venta, consultar identificadores de clientes ya existentes para tener información con sentido real.

En el caso del generador de clientes partimos de la misma lógica, a diferencia de que este solo posee un enumerado para los correos electrónicos de los clientes.

Ambos generadores cuentan con productores de Kafka que se encargan de insertar la información generada en las colas de mensajería, con esto se tiene una mayor eficiencia en el traspaso de la información, tenemos mayor fiabilidad y seguridad. El proceso de los productores se explican en detalles a continuación en las colas de mensajería.

4.2.2. Colas de mensajería

Pruebas de concepto

Como toma de contacto diseñamos dos pruebas de concepto para probar el funcionamiento de los productores y consumidores de Kafka. Como su nombre lo indica los productores son los encargados de insertar la información en las colas de mensajería, en cuanto a los consumidores son los que se encargan de extraer los datos.

Kafka permite crear flujos de datos distintos, estos se denominan *Topics*, estos nos permite mantener nuestra información agrupada en distintos contenedores de flujos de datos.

Para la realización de nuestro prototipo iniciamos por crear un *Topic* de prueba en la consola de comandos.

Entre muchas de las utilidades que nos proporciona Kafka, hacemos énfasis en las siguientes que nos han resultado bastante cómodas para desarrollar.

En primer lugar podemos desplegar Kafka en el servidor local, sin necesidad de alojarlo en un servidor externo o en la nube, esto nos beneficia en el desarrollo y la realización de pruebas. En la implementación de nuestro prototipo creamos un KafkaProducer en lenguaje Python, utilizando la librería “kafka-python”.

Empezamos por configurar el Producer asignándole la ruta *localhost* (Ruta del servidor local) y el puerto donde se aloja el *topic* y posteriormente realizamos la función *send()* que nos permite enviar mensajes al *topic* proporcionado por parámetros.

Esto nos ha servido para experimentar los distintos campos que nos ofrece Kafka como lo son:

- Tiempo de inserción del mensaje.
- Desplazamiento, representa la posición del mensaje en la cola
- Clave, identifica un conjunto de valores
- Valor, representa el contenido del mensaje.

Es importante mencionar que el tiempo de inserción nos sirve a la hora de realizar procesamiento en tiempo real, con esto mantenemos el orden de la generación de los eventos, para mantener la integridad de los resultados. En la figura 4.7 podemos observar el mensaje "Hello, Kafka" introducido en el *topic* denominado "topic-alga". Utilizamos el entorno "Kafka IDE" para visualizar el contenido de las colas de manera gráfica.

Figura 4.7: Primer mensaje de prueba en la cola de mensajería

El segundo prototipo se basa en un Consumidor de Kafka "KafkaConsumer" que realiza lectura los datos introducidos en el *topic* especificado desde el inicio y los imprime en consola. Este nos ayudó a comprender las distintas maneras de leer datos de la cola de mensajería utilizando el atributo *offset*. El desplazamiento en esta aplicación es muy relevante, ya que le indica al consumidor el punto de partida para consumir los datos, bien sea desde el inicio, desde un punto en específico o desde la última lectura realizada. En la figura 4.8 podemos observar los datos en consola que nos proporciona el programa. En este se muestra el mensaje anteriormente mencionado y los atributos de la cola de mensajería.

Figura 4.8: Consumiendo datos de Apache Kafka

Realizar esta serie de programas nos proporcionó los conocimientos necesarios para comenzar a construir nuestra arquitectura.

Tras la finalización de estas dos partes de la arquitectura, damos paso a la evolución del generador de datos anteriormente mencionado, desarrollando en Java (Lenguaje de programación) un "KafkaProducer" a diferencia del anterior este también realiza la tarea de generar los datos sensibles a utilizar en nuestra arquitectura en formato *JSON* y se encarga de introducirlos en la cola de mensajería. Se modificó para realizar la tarea de generar datos de manera constante en un tiempo determinado, con esto conseguimos simular el proceso de creación de nuevos clientes en la empresa. La finalidad de esta versión es manejar la realidad con la que se producen los datos

En una tercera versión de este prototipo se modifica el "KafkaConsumer" para leer datos y almacenarlos en ficheros *JSON*, para su posterior almacenaje en repositorios en la nube.

En la figura 4.9, podemos observar los ficheros de salida que genera nuestra aplicación para su posterior uso en procesamiento *Batch*, almacenando grandes cantidades de datos en ficheros durante periodos largos de tiempo.

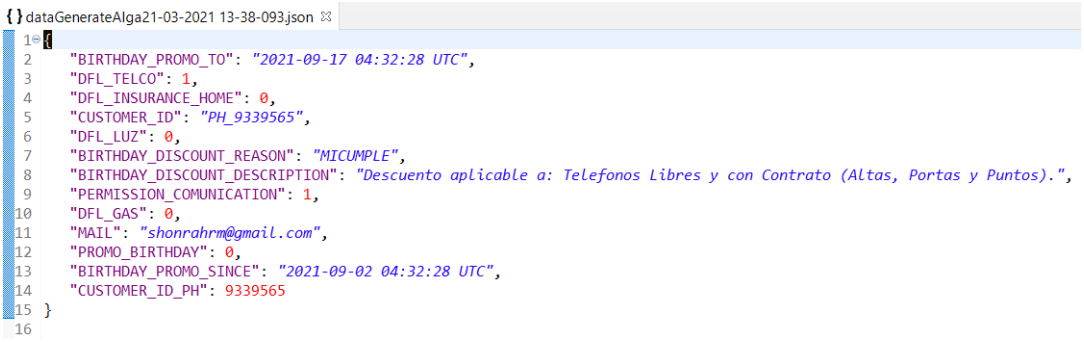


Figura 4.9: Consumiendo datos de Kafka e introduciéndolos en ficheros JSON.

Traspaso de la información

Luego de nuestras pruebas de concepto, damos paso a explicar el proceso de envío de datos de los generadores a las colas de mensajería de Apache Kafka. Este lo implementamos con un proyecto "Maven", ya que nos facilita la descarga de las dependencias necesarias para trabajar con las librerías de esta herramienta. Utilizamos la librería "kafka:2.13, version: 2.6.0" es una de las versiones más recientes y estables.

Para la utilización de las colas de mensajería, en primer lugar, instalamos la versión mencionada anteriormente, como requisito previo se debe tener un entorno de ejecución y desarrollo de Java instalado, ya que Kafka está desarrollado en Scala y Java. Se deben instalar también los servicios de Apache Zoekeeper, este es un servicio centralizado que gestiona los clusters, mensajes y flujos de kafka.

Una vez cumplidos estos requisitos procedemos a iniciar el servicio de zoekeeper. En este caso mostramos el comando necesario para iniciarlo desde el sistema operativo Windows. En consola ejecutamos el siguiente comando:

- *./bin/windows/zookeeper-server-start.bat ./config/zookeeper.properties*

Luego de esto iniciamos la cola de apache kafka:

- *./bin/windows/kafka-server-start.bat ./config/server.properties*

En nuestra implementación utilizamos dos flujos de datos distintos, tantos como generadores tenemos. En uno manejamos la información de los clientes y en el otro la de las ventas. Estos flujos son denominados *topics* y se generan mediante comandos que nos proporciona kafka por consola. Al crear el *topic* debemos pasarle la ruta del servidor con el puerto para que lo pueda localizar. Para este caso hemos trabajado en local, utilizando la ruta (localhost) y puerto por defecto (2181).

- Creación del *topic* para el generador de clientes:
  ./bin/windows/kafka-topics.bat – create – zookeeper localhost:2181 – replication-factor 1 – partitions 1 – topic customer-details-topic

- Creación del *topic* para el generador de ventas:
  ./bin/windows/kafka-topics.bat – create – zookeeper localhost:2181 – replication-factor 1 – partitions 1 – topic sales-details-topic

Una vez generada la información y creados los *topics* desarrollamos el *producer*, siguiendo los siguientes pasos:

- Utilizamos las librerías de Apache Kafka para instanciar objetos de la clase *KafkaProducer* donde le debemos indicar el servidor en el que se va a alojar y un par clave/valor predeterminado.
- A continuación añadimos un atributo de la clase *ProducerRecord*. Este es una grabación de datos que nos permite indicarle el nombre del *topic* creado, donde se almacenarán los mensajes y a su vez se insertan los datos a enviar.
- Realizamos la llamada al método *send()* de la clase *KafkaProducer* y le pasamos la grabación de datos anteriormente mencionada. Finalmente nos queda cerrar la conexión del productor haciendo una llamada al método *close()*.

Este proceso envía la información proveniente de los generadores de datos de manera indefinida con una latencia de un minuto entre envío y envío. Con esto simulamos las altas de nuevos clientes y las ventas realizadas en tiempo real en cualquier tienda The Phone House en España.

4.2.3. Capa Real Time

Procesamiento

Este proceso consiste en los análisis, transformaciones y actualizaciones incrementales de los datos en el mismo momento que se van recibiendo por Apache Kafka buscando los resultados que se desean. Hemos seleccionado la herramienta de Apache Spark debido a la gran cantidad de funcionalidades que tiene para trabajar y manejar la información tanto el los procesos de análisis en tiempo real como con grandes lotes de información.

En este proceso de la arquitectura, seleccionamos y concretamos la información exacta que queremos visualizar en un futuro. Esta selección se realiza mediante consultas, en un intervalo de tiempo determinado, a la información que nos llega de Apache Kafka. Los resultados son insertados en tablas SQL creadas para estos datos concretos y que tienen porque corresponderse con los campos de la información recibida en un primer momento.

En nuestros procesos de análisis, utilizamos una funcionalidad de Apache Spark denominada *window*. Esto nos permite, dado un periodo de tiempo, además de extraer la parte de la información deseada también tiene en cuenta el valor de los datos más recientes, actualizando la información. Esto lo convierte como una de las funcionalidades más utilizadas en el proceso de flujo de datos. Además, con el deslizamiento de la ventana, es decir, cambio del periodo de tiempo, los elementos antiguos del flujo de datos se eliminan de dicha ventana y los elementos recién llegados se insertan en la ventana para su procesamiento ((Xiao and Hu, 2020)).

Por otro lado, indicamos con *withWatermark()* cuando cerrar la ventana temporal de eventos, consiguiendo que si viene un dato con la hora de otra ventana anterior permite actualizar esa ventana con ese nuevo dato, en vez de borrarlo. Además, los operadores

permiten a los usuarios escribir una lógica personalizada para implementar el análisis de la información de estas ventanas como deseen ((Armbrust, Das, Torres, Yavuz, Zhu, Xin, Ghodsi, Stoica, and Zaharia, 2018)). Este proceso de análisis se tiene que llevar a cabo con el departamento o cliente al que va dirigido esta parte de la arquitectura ya que estos análisis dependen de lo que se esté necesitando visualizar. En nuestro caso, utilizando estas funcionalidades para calcular el número de las ventas realizadas en cada ciudad de España, la suma del beneficio de las ventas junto con la suma de los descuentos aplicados por producto y la diferencia de ventas que existen en cada hora.

```
+------------------------------------------+----------------------+-----+
|window                                    |SHOP_CITY             |count|
+------------------------------------------+----------------------+-----+
|[2021-06-07 17:00:00, 2021-06-07 18:00:00]|GIJON                 |2    |
|[2021-06-07 17:00:00, 2021-06-07 18:00:00]|IBIZA                 |3    |
|[2021-06-07 17:00:00, 2021-06-07 18:00:00]|SEVILLA               |9    |
|[2021-06-07 17:00:00, 2021-06-07 18:00:00]|MALAGA                |2    |
|[2021-06-07 17:00:00, 2021-06-07 18:00:00]|SANTANDER             |7    |
|[2021-06-07 17:00:00, 2021-06-07 18:00:00]|BARCELONA             |6    |
|[2021-06-07 17:00:00, 2021-06-07 18:00:00]|VITORIA               |2    |
|[2021-06-07 17:00:00, 2021-06-07 18:00:00]|PALENCIA              |4    |
|[2021-06-07 17:00:00, 2021-06-07 18:00:00]|CUENCA                |2    |
|[2021-06-07 17:00:00, 2021-06-07 18:00:00]|SANTIAGO DE COMPOSTELA|1    |
|[2021-06-07 17:00:00, 2021-06-07 18:00:00]|CORDOBA               |3    |
|[2021-06-07 17:00:00, 2021-06-07 18:00:00]|CACERES               |2    |
|[2021-06-07 17:00:00, 2021-06-07 18:00:00]|GUADALAJARA           |2    |
|[2021-06-07 17:00:00, 2021-06-07 18:00:00]|SANTA CRUZ DE TENERIFE|1    |
|[2021-06-07 17:00:00, 2021-06-07 18:00:00]|MURCIA                |3    |
|[2021-06-07 17:00:00, 2021-06-07 18:00:00]|ZAMORA                |2    |
|[2021-06-07 17:00:00, 2021-06-07 18:00:00]|ALBACETE              |2    |
|[2021-06-07 17:00:00, 2021-06-07 18:00:00]|LARIOJA               |2    |
|[2021-06-07 17:00:00, 2021-06-07 18:00:00]|MADRID                |4    |
|[2021-06-07 17:00:00, 2021-06-07 18:00:00]|CARTAGENA             |2    |
+------------------------------------------+----------------------+-----+
```

Figura 4.10: Resultados del número de ventas por ciudad

En el proceso de las ventas por ciudades y teniendo como ventana de tiempo de 17:00 a 18:00 como se observa en la fotografía 4.10 vamos a suponer que en la ciudad de Madrid se realiza una venta a las 17:10. Se inserta como resultado en la tabla, Madrid a "1". A las 17:30 se realizan dos ventas en Madrid por lo que el registro de Madrid a "1" es actualizado para insertar Madrid a "3" y así sucesivamente con todas las ciudades en donde al menos se ha realizado una venta hasta las 18:00. A partir de ese momento el contador se volvería a poner a "0" ya que empezaría una nueva ventana de tiempo. De esta manera, obtenemos como resultado una tabla de cada ciudad con su número de ventas durante cada hora. En este ejemplo agrupamos por ciudades mediante la función *groupby()* donde pasamos como parámetros la ventana de tiempo por un lado y el campo deseado por otro:

```
window(df.CLOSED_DATE, '1 hour', '1 hour'), df.SHOP_CITY
```

Para realizar la cuenta de esto utilizamos la función *count()*.

A partir de esta idea, se pueden hacer un gran número de consultas de cualquier grado de dificultad con un número de operaciones, agrupaciones y comparaciones de cualquier tipo para obtener los resultados en las columnas deseadas. Para ello Apache Spark ofrece una variada cantidad de funcionalidades con las que trabajar y que pueden llegar a ser muy complicadas si no se conoce previamente su documentación. Por ejemplo, en el proceso

de obtención de la suma de los beneficios y del dinero descontado según el producto y el tipo de descuento hemos aumentado el número de campos agrupados,

```
.groupBy(window(df.CLOSED_DATE, '1 hour', '1 hour'),
df.FAMILIA_LTV, df.DISCOUNT_REASON)
```

utilizando también la función *sum()* de los campos correspondientes:

```
.sum('TOTAL_DISCOUNT_AMOUNT', 'TOTAL_RETAIL_AMOUNT')
```

Obteniendo los resultados de la fotografía 4.11.

window	FAMILIA_LTV	DISCOUNT_REASON	sum(TOTAL_DISCOUNT_AMOUNT)	sum(TOTAL_RETAIL_AMOUNT)
[2021-06-07 17:00:00, 2021-06-07 18:00:00]	SEGUROS	DTOPROM30	45.0	105.0
[2021-06-07 17:00:00, 2021-06-07 18:00:00]	DUPLICADOS	EMPLEADOS	0.5	4.5
[2021-06-07 17:00:00, 2021-06-07 18:00:00]	POSTPAY_YOIGO	MICUMPLE	70.0	630.0
[2021-06-07 17:00:00, 2021-06-07 18:00:00]	POSTPAY_ORANGE	EMPLEADOS	50.0	450.0
[2021-06-07 17:00:00, 2021-06-07 18:00:00]	PC/TABLET	WEB	45.0	855.0
[2021-06-07 17:00:00, 2021-06-07 18:00:00]	LIBRE	MICUMPLE	180.0	1620.0
[2021-06-07 17:00:00, 2021-06-07 18:00:00]	SERVICIOS	DTOPROM30	7.5	17.5
[2021-06-07 17:00:00, 2021-06-07 18:00:00]	SEGUROS	SUPERCIERRE	9.0	51.0
[2021-06-07 17:00:00, 2021-06-07 18:00:00]	LIBRE	WEB	90.0	1710.0
[2021-06-07 17:00:00, 2021-06-07 18:00:00]	POSTPAY_ORANGE	DTOPROM30	120.0	280.0
[2021-06-07 17:00:00, 2021-06-07 18:00:00]	POSTPAY_YOIGO	EMPLEADOS	30.0	270.0
[2021-06-07 17:00:00, 2021-06-07 18:00:00]	POSTPAY_VODAFONE	EMPLEADOS	40.0	360.0
[2021-06-07 17:00:00, 2021-06-07 18:00:00]	DUPLICADOS	DTOPROM30	3.0	7.0
[2021-06-07 17:00:00, 2021-06-07 18:00:00]	SERVICIOS	WEB	0.75	14.25
[2021-06-07 17:00:00, 2021-06-07 18:00:00]	PC/TABLET	DTOPROM30	360.0	840.0
[2021-06-07 17:00:00, 2021-06-07 18:00:00]	PC/TABLET	MICUMPLE	30.0	270.0
[2021-06-07 17:00:00, 2021-06-07 18:00:00]	LIBRE	DTOPROM30	720.0	1680.0
[2021-06-07 17:00:00, 2021-06-07 18:00:00]	POSTPAY_YOIGO	WEB	10.0	190.0
[2021-06-07 17:00:00, 2021-06-07 18:00:00]	DUPLICADOS	WEB	0.75	14.25
[2021-06-07 17:00:00, 2021-06-07 18:00:00]	POSTPAY_ORANGE	SUPERCIERRE	15.0	85.0

Figura 4.11: Resultados del beneficio y del descuento agrupado por producto y tipo de descuento

En un tercer proceso, utilizamos también la función *sum()* para obtener el beneficio total de todas las tiendas de España conseguido en la hora actual como se puede observar en la fotografía 4.12 para posteriormente realizar una comparación con el resto de horas en la parte de visualización ya que en la base de datos dispondremos de los resultados de las horas anteriores.

window	sum
[2021-06-07 17:00:00, 2021-06-07 18:00:00]	14007.5

Figura 4.12: Resultado del beneficio total por hora

Guardado de datos en SQL Server

En un primer momento este servicio de almacenamiento solo iba a ser utilizado para guardar una parte de la información recibida de la empresa. Se iba a guardar únicamente

la parte de la información seleccionada y analizada mediante las consultas en realizadas en el procesamiento de los datos provenientes de Apache Kafka y que posteriormente se mostrarán por el *dashboard*. Por otro lado, sería MongoDB la herramienta elegida para volcar toda la información completa de la empresa sin ningún tipo de modificación haciendo de esta como almacenamiento del histórico de datos. Debido al abandono del integrante del grupo encargado de realizar el estudio de esta herramienta y sus correspondientes conexiones con Apache Kafka nos vimos obligados a utilizar este servicio además de como almacenamiento de la información analizada en las consultas, como almacenamiento de toda la información de la empresa, en dos tablas SQL con los mismos campos que los generadores.

Hemos creado un servidor de bases de datos Azure SQL Server en el que almacenamos de manera relacional los datos modificados y seleccionados provenientes de los procesos de análisis en tiempo real. Este guardado se realiza en distintas tablas para facilitar el acceso y la selección de estos datos de la manera más sencilla posible para su análisis posterior. Para la inserción de esta información a la base de datos de Microsoft Azure hemos elaborado un programa en Python utilizando *PySpark* y con su módulo de conexión *pyodbc*.

Para ello, hemos realizado los siguientes pasos:

- Creación del *consumer*. Accedemos al extremo de Apache Kafka que recibe la información del *producer* utilizando su mismo *topic*. Para este paso, hemos utilizado la herramienta Apache Spark ya que nos permite un manejo muy cómodo de la información que vamos recibiendo cada pocos segundos. En nuestro caso, antes de guardar la información recibida la analizamos mediante consultas que veremos más adelante. Para guardar información proveniente de Apache Kafka, primero creamos una sesión con Apache Spark, luego, vamos leyendo la información en *streaming* con la función *readStream* indicando tanto el *topic* como la dirección de red a la cual nos vamos a conectar.
- Creamos un esquema de la información que esperamos recibir mediante el método *StructType* con los nombres de los parámetros y el tipo de dato que son.
- Transformamos los *JSONs* recibidos de Apache Kafka en datos relacionales mediante el esquema diseñado en el punto anterior utilizando el método *from_json()*. Esto nos devolverá un *dataframe(df)* con la información correspondiente.
- Descargamos e instalamos el driver *ODBC Driver 13 for SQL Server* para la posterior conexión con la base de datos y creamos una variable con la cadena de conexión necesaria para el acceso a la base de datos.
- Escritura del *dataframe* con la información mediante la función *writeStream*. Como estamos recibiendo lotes de información cada pocos segundos, abrir y cerrar la conexión con el servidor de Azure por cada lote supondría un coste muy elevado y una perdida de tiempo considerable al ser una arquitectura diseñada para gran cantidad de lotes. Es por eso, que empleamos también las funciones *forEachBatch()* y *triger()*. La primera nos permite almacenar un grupo mayor de lotes, en vez de uno en uno, que serán tratados en una función auxiliar añadida como parámetro, en nuestro caso "foreach_batch_function()". La segunda le indicamos el tiempo de espera mientras se van añadiendo dichos lotes, en nuestro caso 30 segundos.

Añadimos también las funciones *start()* y *awaitTermination()* ya que al estar trabajando con datos en *streaming* necesitamos indicar cuando empezamos y terminamos.

- Implementación de la función "foreach_batch_function" en la que insertamos los datos en la tabla correspondiente de la base de datos. En primer lugar, con las funciones *select()* y *collect()* volcamos la información recibida a una nueva variable que la llamamos *data* para tener acceso a ella mediante filas, que es como se va a insertar en las tablas SQL. En segundo lugar, iniciamos la conexión con Azure, mediante *pyodbc.connect* y la variable con la cadena de conexión al servidor, previamente indicada. En tercer lugar comenzamos el bucle para recorrer fila a fila los datos de variable *data* con un bucle *for row in data*. En tercer lugar, como el campo *window*, en las filas, es una estructura con la fecha de inicio y final de la venta juntas en un mismo campo como se puede ver en cualquiera de las tres fotografías anteriores, necesitamos separarlos para insertarlos en campos distintos en la base de datos mediante *windowTime = row['window']*. A continuación, hacemos un borrado de esa fila si ya se encuentran en la tabla para volver a insertarlos pero actualizados. Por último insertamos en la tabla el nuevo dato ejecutando la operación *insert* en SQL y añadiendo los campos del *dataframe* a la tabla con sus respectivos valores. Una vez recorridas todas las filas, cerramos la conexión con la base de datos.

4.2.4. Capa Batch

La capa *Batch* de nuestra arquitectura empieza desde un consumidor que se encarga de extraer los datos de todo un día de las colas de mensajería. Luego este realiza un volcado de la información en el repositorio de Azure Storage, este repositorio tiene el histórico de todos los datos. Posteriormente tenemos un proceso que coge los datos por día del repositorio y se encarga de procesarlos para realizar las consultas requeridas. Este mismo proceso genera una serie de informes debido a que este procesamiento se hace de manera distribuida. Finalmente existe otro proceso que se encarga de coger los informes distribuidos y generar un solo informe por consulta realizada. Estos informes finales sirven para ser analizados.

Luego del resumen del funcionamiento general de la capa *Batch*, damos paso a explicar detalladamente cada proceso.

Guardado de datos en Azure Storage

Desplegamos una instancia de Azure Blob Storage como *master data-set* en los servidores Cloud de Microsoft Azure. Este nos permite almacenar grandes cantidades de datos sin necesidad de estructuración de tablas relacionales. Allí almacenamos todos datos de las fuentes de generación de datos, las actualizaciones de los mismos generados por programas propios que simulan el proceso de compras en las tiendas y todos los cambios posibles que puedan surgir en el procesamiento de los datos.

Como requisito previo, la herramienta de Azure Storage nos permite crear contenedores, en estos podemos almacenar información y clasificarla de la manera deseada. En nuestro caso, utilizamos un solo contenedor en el que se clasificará internamente en una estructura de tres directorios.

En el directorio *Raw* almacenamos la información exactamente igual a la que consumimos

de las colas de mensajería, sin realizar ninguna modificación sobre ellas. Con esto, podemos dividir la información sin procesar por un lado y la información procesada, denominada información de negocio, por otro, que será explotada posteriormente en informes que descargará el equipo de negocio. En la figura 4.13, podemos observar el contenido de los detalles de ventas de todas las tiendas el día 13/06/2021 en un fichero JSON en el directorio *Raw*.

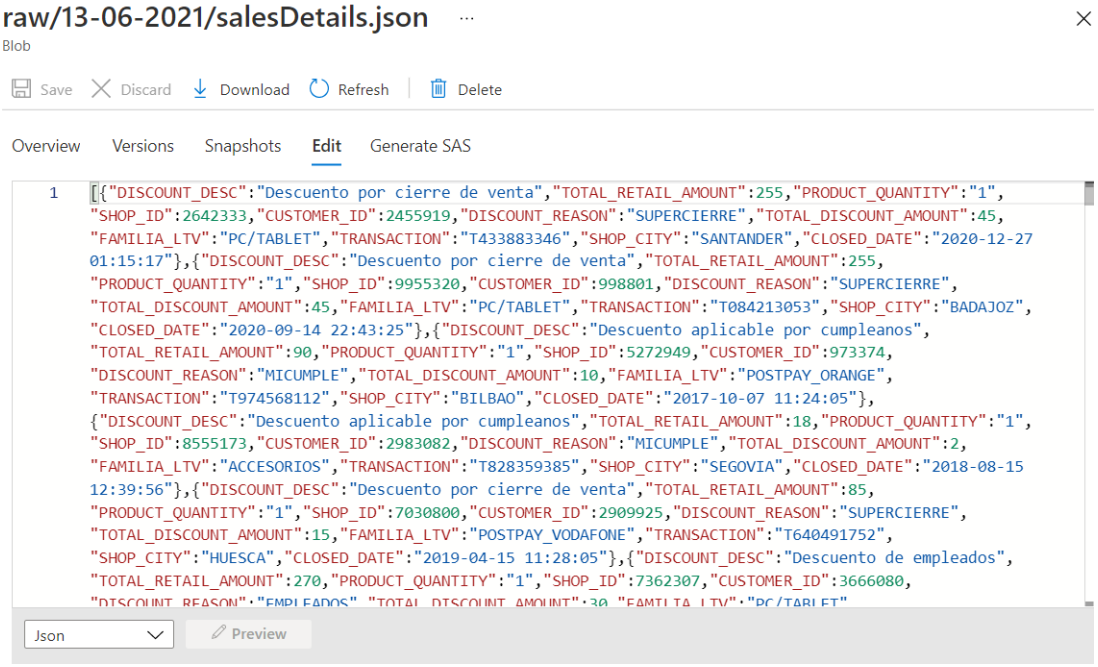


Figura 4.13: Fichero con detalles de ventas del día 13/06/2021

En el directorio *Common* almacenaremos los datos procesados, es decir una vez leídos todos ficheros con extensión *JSON* incluidos en el directorio *Raw* realizaremos una serie de procesos y consultas de los mismos, este proceso lo comentaremos con mejor detalle en la sección de procesamiento *Batch*. La información contenida en este directorio ya contiene información válida para negocio, pero no posee la estructura deseada para ser representada.

El directorio *Bussines* contiene los datos con valor y significados justos para realizar su explotación, es decir, los informes con el formato adecuado que negocio puede descargar y realizar sus análisis deseados.

Volcado de datos

La inyección de los datos en nuestro *Storage* la realizamos mediante una *API REST* implementado con el *framework* de Spring Boot en Java. Como ventaja de trabajar con Spring, tenemos que nos facilita crear esta aplicación levantándola como micro-servicio, evitando tener que alojar el programa en un servidor de aplicaciones. Aunque para el caso de nuestro proyecto hemos trabajado de manera local, trabajar de esta manera nos proporciona una mayor escalabilidad a la hora de implementar la arquitectura en un entorno de producción real. También podemos mencionar que minimiza el consumo de recursos al tener esta arquitectura basada en micro-servicios ya que se despliegan sólo las funcionalidades necesarias.

La *API REST* se despliega en un servidor de apache llamado Tomcat que nos proporciona Spring, este mantiene disponible los servicios para cuando sea necesario consultar o modificar los datos en nuestro contenedor.

Utilizamos como referencia la documentación de Microsoft Azure para construir nuestro programa. ((Azure, 2021)).

En nuestra *API REST* utilizamos las siguientes dependencias Maven de Azure para tener acceso a las librerías que contienen los servicios de acceso a los *blobs* de Azure Storage necesarios: "spring-starter-azure-storage, versión 1.2.8".

Entre las ventajas de usar un proyecto en Spring, tenemos un "inicializador" de proyectos que nos ofrece la propia plataforma ((VMware, 2013-2021)). Esta herramienta nos permite configurar de manera muy sencilla la versión de Java a utilizar, agregar dependencias de manera rápida, elegir el empaquetado de la aplicación junto con otros elementos básicos para empezar a desarrollar.

Luego generamos un fichero de autenticación con los comandos que nos proporciona Microsoft Azure. Este contiene las credenciales de acceso al servidor remoto donde se aloja el contenedor.

Una vez realizados estos pasos previos, añadimos la configuración necesaria de acceso a las propiedades del proyecto, es importante destacar que entre los datos necesarios debemos indicar como datos obligatorios los siguientes:

- Fichero de autenticación anteriormente mencionado.
- Nombre de la cuenta de Azure Storage.
- Nombre de contenedor.

Con esto, al ejecutar la aplicación, se inyectan toda las propiedades y dependencias necesarias para empezar a desarrollar nuestra *API REST*.

Creamos la clase principal para ejecutar la aplicación:

- En el controlador, definimos los métodos que nos permiten realizar las operaciones de lectura y escritura en el contenedor. Con la etiqueta *RequestMapping* definimos el nombre del servicio de la *API REST*. Tenemos un atributo *resource* que identificará el fichero sobre el cual queremos escribir o leer la información. El valor del recurso se define en el fichero de configuración de las propiedades.
- Con este servicio ejecutándose en el servidor Tomcat, solo debemos ejecutar peticiones *POST* a la *URL* del servicio de la *API REST* para enviar información al contenedor ó en caso de querer obtener datos se realiza una petición *GET*. Estas peticiones son del protocolo *HTTP*.
- Para consumir los datos provenientes de las colas de mensajería, desarrollamos un Consumidor de Apache Kafka, Que se está ejecutando a la espera de que llegue información en los flujos de datos indicados. Como tenemos dos flujos distintos, hemos desarrollado dos consumidores. A diferencia de los productores que mencionamos en apartados anteriores, en el caso de consumir datos, debemos configurar las propiedades de Apache Kafka para "deserializar" los pares "clave/valor" de las grabaciones realizadas por los productores en el otro extremo de la cola de mensajería.

También debemos indicar el desplazamiento desde el que se quiere empezar a consumir. En nuestro caso empezamos consumiendo lo que hay desde el principio, para que luego el consumidor se quede a la espera de nuevos datos.

- Al consumir las grabaciones de datos proveniente de los flujos de datos (topics) hacemos una llamada al método *POST* con la *URL* correspondiente del servicio del controlador del *API REST*. Finalmente con esta llamada logramos introducir ficheros *JSON* con la información dentro de la capa RAW en Azure Storage.

Procesamiento de los datos

A diferencia de la capa de *Real Time*, los datos en la capa *Batch* se procesan después de haber sido almacenados pero también utilizamos Apache Spark para su desarrollo y lograr nuestros objetivos. El procesamiento de todos los datos que pasan por nuestra arquitectura se encuentran en el repositorio de Azure Blob Storage explicado anteriormente. Estos están almacenados en ficheros *JSON* como grandes cantidades de datos que no son fijas, ya que dependen de los eventos que se van generando en la simulación de la llegada de nuevos clientes y la realización de ventas. Debido a esto, hemos desarrollado un programa con la tecnología Scala Spark, para su procesamiento. Utilizamos Scala en este caso para tener una alternativa más en el procesamiento ya que la capa de *Real Time* la realizamos con PySpark. Aunque el procesamiento en ambas capas es muy similar debido a las ventajas que nos proporciona Spark, en este caso tenemos que realizar lectura de una manera óptima para poder procesar lotes muy grandes de datos.

En primera instancia para trabajar con Spark definimos una sesión, en dicha sesión especificamos el nombre del proceso y el cluster en el que se va a desplegar nuestro programa. Entre las alternativas de despliegue que nos ofrece Spark, elegimos el cluster independiente de Spark. Este modo de depliegue utiliza el modelo "Master/Slave" que consiste en un nodo maestro con el proceso principal y varios nodos esclavos que se denominan *Workers* los cuales se encargan de realizar el procesamiento de cada tarea que se les asigna. En este proceso participa un *Driver* que se encarga de distribuir las distintas actividades a realizar por cada *worker*. Este proceso distribuido Spark lo denomina como RDD (Resilient Distributed Dataset).

Como mencionamos en el capítulo de "Estado del arte" Spark se apoya en Hadoop para el procesamiento *Batch*. Ya configurada nuestra sesión de Spark, procedemos a modificar la configuración de Hadoop a nuestra conexión abierta al nodo de Spark (Sparkcontext) configurada anteriormente. En dicha configuración indicamos el método escogido para acceder a la conexión con Azure, que en este caso es a través de un sistema de archivos que nos permite la realización de operaciones de lectura y escritura en los archivos que se denomina *NativeAzureFileSystem*. A continuación, le proporcionamos a la configuración la clave secreta que se genera en el portal de Azure para acceder al Azure Storage.

Una vez realizados los pasos de configuración, definimos dos esquemas para maquetar la información leída en estructuras mejor conocidas por nosotros y con ello poder manejar estas estructuras de una manera más cómoda. Definimos uno para los datos del cliente y otro para el de ventas.

La tarea de leer los datos se realiza de manera distribuida por los trabajadores que mencionamos anteriormente, Spark nos permite de manera sencilla leer todos los datos contenidos en un directorio especificado. Esta opción nos da una ventaja enorme ya que

como mencionamos anteriormente los ficheros se van almacenando en el directorio *Raw*. Realizamos el proceso de lectura haciendo una llamada al método *read()* de Spark, este realiza la lectura de todos los ficheros del directorio especificado, distribuyéndolos en cada uno de los nodos trabajadores. En el proceso de lectura le indicamos opciones para leer múltiples líneas, esto se debe a que la información almacenada está almacenada en múltiples líneas. También le indicamos la opción de incluir las cabeceras de los datos, para extraer todo el contenido de los ficheros. Para finalizar el proceso de lectura le indicamos el formato en el que se quiere realizar dicha lectura que en este caso es *JSON*.

Al tener nuestros datos ya extraídos, creamos una tabla temporal SQL llamada *sales* con el contenido de los mismos. Esta tabla nos permite realizar las consultas de una manera más cómoda, a diferencia de tener que tratar con datos expresados en formato *JSON*.

Sobre la tabla temporal *sales* realizamos las siguientes operaciones de consultas SQL:

- Acumulado total de ingresos por ventas del día a nivel de todas las tiendas de la empresa.
- Acumulado del número de ventas realizadas por ciudad.
- Acumulado de los ingresos por producto y razón de descuento del producto. Acumulado del monto total descontado por producto y razón de descuento.

Capítulo 5

RESULTADOS

5.1. Visualización capa Real Time

Una vez llevados a cabo los procesos de selección y análisis de la información, como se puede observar en el ejemplo explicado de las ventas de cada ciudad en cada hora, e insertados en sus correspondientes tablas SQL, llega el momento de mostrar estos resultados en una herramienta de visualización para poder obtener conclusiones sobre ellos.

En primer lugar, establecemos la conexión con la base de datos de donde obtener los datos para ser mostrados. Grafana ofrece una gran cantidad de conexiones con múltiples bases de datos donde se encuentra SQL Server de Azure. Indicamos el nombre de nuestra base de datos junto con el *host* de acceso a los datos y las credenciales de usuario y contraseña de nuestro servidor.

A continuación, creamos un *dashboard* donde añadimos los respectivos paneles que van a mostrar la información de los distintos procesos que tengamos en ejecución mediante gráficas. Los paneles nos ofrecen, multitud de formas y configuraciones para visualizar y manejar la información según la queramos exponer. Esa información se extrae de la base de datos con consultas SQL donde el tiempo de las ventanas explicadas anteriormente, adquiere un papel muy importante ya que se va a mostrar la información según ha ocurrido en el tiempo. Seleccionamos la opción de *time series* para que la información se vaya modificando a medida que pase el tiempo y muestre los datos que vayan llegando en referencia a las ventanas.

Las consultas SQL aceptan los mismos parámetros y la misma sintaxis que si se realiza en la propia base de datos a la que te conectas, en este caso a SQL Server de Azure con la única diferencia del campo *select*. Este campo tiene que tener tres parámetros con sus respectivos valores. Por un lado, el parámetro *time* haciendo referencia a la columna que contiene las ventanas de tiempo de la información que se va a mostrar. Por otro lado el parámetro *metric* indicando el campo del que se van a ver los cambios con el paso del tiempo y por último el parámetro *value* los valores que van a quedar reflejados con respecto al parámetro *metric*. En el campo del *where*, Grafana nos permite poner variables globales que pueden afectar a todos paneles del *dashboard* como explicaremos más adelante con un ejemplo. Para ello, se tienen que crear en la configuración del *dash-*

board en el apartado de variables, donde creas una consulta que devuelva los valores que deseamos que se seleccionen en un futuro.

En nuestro caso para mostrar una comparación del número de ventas de distintas ciudades seleccionamos la opción de gráfica e indicamos en el campo *select* los parametros *TIMESTART as time*, *CUENTA as value* y *SHOP_CITY as metric*. El *from* sería exactamente igual que en la base de datos, *dbo.SALESBYCITY* y en el *where*, como en este caso tenemos una gran cantidad de ciudades y queremos ver solo las seleccionadas por esas variables globales lo indicamos como *SHOP_CITY IN ($CITY)* donde *$CITY* es la variable global que nos devuelve todas las ciudades que queramos comparar. Por último, ordenamos según el tiempo con la función *orderBy()* en orden descendente.

Figura 5.1: Resultados del número de ventas de las ciudades seleccionadas

Para mostrar las comparación del beneficio con el dinero descontado por cada descuento según el producto seleccionamos una gráfica de barras. En el campo *select* indicamos los parámetros *TIMESTART as time*, *PRICEAMOUNT*, *DISCOUNTAMOUNT as value* para visualizar la diferencia entre estos dos campos y *DISCOUNT_REASON as metric*. En el *where* indicamos *FAMILIA_LTV IN ($PRODUCT)* donde *$PRODUCT"* es la variable global que selecciona el producto del que queremos comparar todos los descuentos ofrecidos.

Figura 5.2: Diferencias entre el beneficio obtenido y el dinero descontando por producto seleccionado con cada promoción

En nuestro último panel, mostramos también una gráfica de barras directamente con la diferencia entre las ventas producidas entre cualquier hora y su anterior. Para ello utilizamos la función *lag()* que se implementaría igual que en la base de datos de Azure. En el campo *select* indicamos en primer lugar *TIMESTART as time* y a continuación

AMOUNT - LAG(AMOUNT, 1) OVER (ORDER BY TIMESTART ASC)

donde se devuelve la diferencia del beneficio actual menos el beneficio de la hora anterior y ordenado por las ventanas de tiempo de menor a mayor.

Figura 5.3: Resultados la diferencia de beneficios con respecto a la hora anterior

Una vez insertadas estos paneles en el *dashboard*, se colocan de la manera deseando dando a cada una de ellas el tamaño óptimo para su correcta visualización. Desde esta pantalla general, se pueden pueden modificar los valores de las ventanas de *CITY* y *PRODUCT* observado los cambios en los valores de todas las tablas a las que hacen

referencia. También se puede interactuar con el rango de tiempo del que se quieren mostrar los datos.

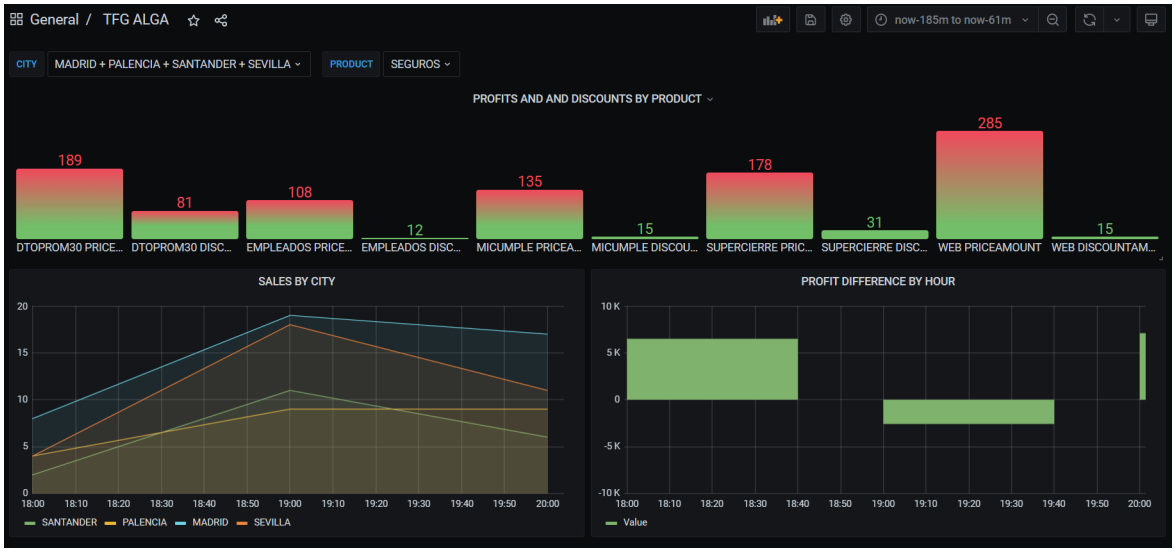


Figura 5.4: Visualización definitiva del *dashboard*

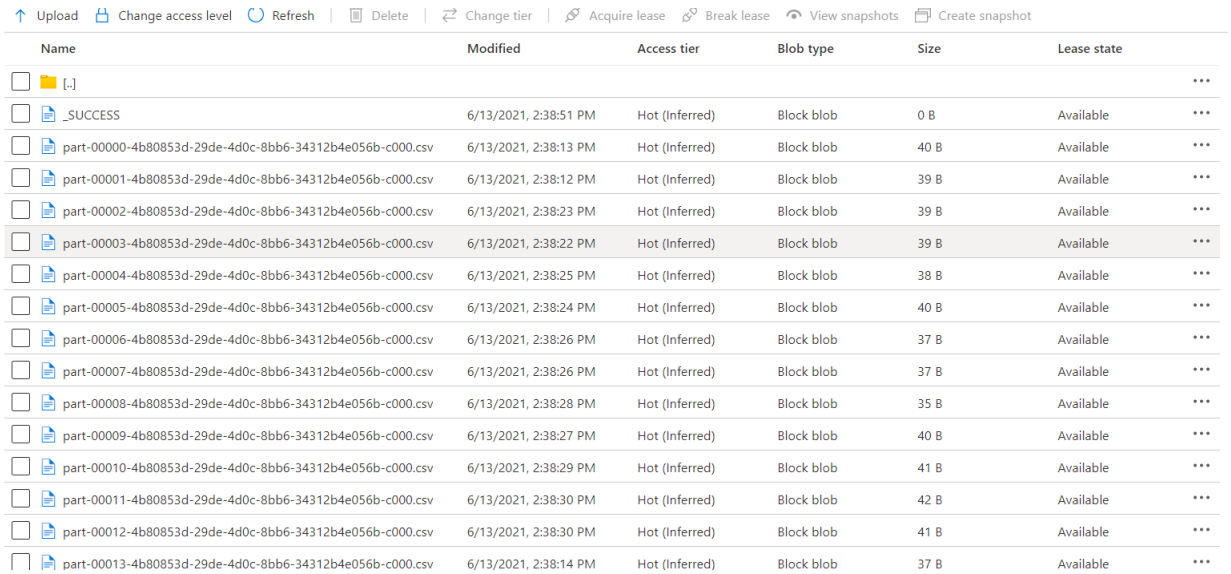
5.2. Explotación capa Batch

Los resultados obtenidos de las consultas, los insertamos en ficheros con formato Csv en el directorio *Common*, la información en este directorio se estructura a su vez en un directorio que lleva por nombre la fecha actual de cada día que se extrae dicho informe. Estos ficheros por día se almacenan en un directorio con el nombre que identifica la consulta realizada.

En la escritura de los resultados, delimitamos los datos por el símbolo de ",". Incluimos los campos de cabecera con un renombrado especificado, con esto indicamos el significado de los datos representados en el fichero. También indicamos la codificación (UTF-8) en la escritura.

Al terminar el procesamiento de los datos mostramos por consola el tiempo de duración del proceso de inicio a fin para poder realizar comparaciones de eficiencia.

No obstante, a la hora de escribir la información de manera distribuida, Spark genera un fichero de salida por cada *worker* que realiza el proceso. En la figura 5.5, podemos observar el contenido del directorio *Common* con los ficheros distribuidos, resultados del procesamiento.



Name	Modified	Access tier	Blob type	Size	Lease state
[..]					
_SUCCESS	6/13/2021, 2:38:51 PM	Hot (Inferred)	Block blob	0 B	Available
part-00000-4b80853d-29de-4d0c-8bb6-34312b4e056b-c000.csv	6/13/2021, 2:38:13 PM	Hot (Inferred)	Block blob	40 B	Available
part-00001-4b80853d-29de-4d0c-8bb6-34312b4e056b-c000.csv	6/13/2021, 2:38:12 PM	Hot (Inferred)	Block blob	39 B	Available
part-00002-4b80853d-29de-4d0c-8bb6-34312b4e056b-c000.csv	6/13/2021, 2:38:23 PM	Hot (Inferred)	Block blob	39 B	Available
part-00003-4b80853d-29de-4d0c-8bb6-34312b4e056b-c000.csv	6/13/2021, 2:38:22 PM	Hot (Inferred)	Block blob	39 B	Available
part-00004-4b80853d-29de-4d0c-8bb6-34312b4e056b-c000.csv	6/13/2021, 2:38:25 PM	Hot (Inferred)	Block blob	38 B	Available
part-00005-4b80853d-29de-4d0c-8bb6-34312b4e056b-c000.csv	6/13/2021, 2:38:24 PM	Hot (Inferred)	Block blob	40 B	Available
part-00006-4b80853d-29de-4d0c-8bb6-34312b4e056b-c000.csv	6/13/2021, 2:38:26 PM	Hot (Inferred)	Block blob	37 B	Available
part-00007-4b80853d-29de-4d0c-8bb6-34312b4e056b-c000.csv	6/13/2021, 2:38:26 PM	Hot (Inferred)	Block blob	37 B	Available
part-00008-4b80853d-29de-4d0c-8bb6-34312b4e056b-c000.csv	6/13/2021, 2:38:28 PM	Hot (Inferred)	Block blob	35 B	Available
part-00009-4b80853d-29de-4d0c-8bb6-34312b4e056b-c000.csv	6/13/2021, 2:38:27 PM	Hot (Inferred)	Block blob	40 B	Available
part-00010-4b80853d-29de-4d0c-8bb6-34312b4e056b-c000.csv	6/13/2021, 2:38:29 PM	Hot (Inferred)	Block blob	41 B	Available
part-00011-4b80853d-29de-4d0c-8bb6-34312b4e056b-c000.csv	6/13/2021, 2:38:30 PM	Hot (Inferred)	Block blob	42 B	Available
part-00012-4b80853d-29de-4d0c-8bb6-34312b4e056b-c000.csv	6/13/2021, 2:38:30 PM	Hot (Inferred)	Block blob	41 B	Available
part-00013-4b80853d-29de-4d0c-8bb6-34312b4e056b-c000.csv	6/13/2021, 2:38:14 PM	Hot (Inferred)	Block blob	37 B	Available

Figura 5.5: Contenido del directorio *Common*

Esto no tiene mucho sentido a la hora de visualizar los informes ya que el contenido estará distribuido en varios ficheros distintos. Para esto utilizamos el directorio *Common* como paso intermedio al representar la información a negocio. Aunque ya en este paso representa información de valor, no es la versión definitiva.

Como solución a esto desarrollamos otra aplicación en Scala Spark igualmente, que se encarga de leer los ficheros por cada día en el directorio *Common/nombre de la consulta/fecha actual*. Este realiza el proceso de concatenación de la información para generar un único fichero de salida en el directorio *Business* representable para ser analizado por el equipo correspondiente.

La concatenación se realiza juntando en una sola partición los resultados de los informes por día, con la función *repartition()*. Nos limitamos a utilizar esta función solo para estos casos que no se debe procesar demasiada información ya que esta une en un sola partición el procesamiento de todos los trabajadores, ocasionando un pésimo rendimiento en aplicaciones distribuidas de *Big Data*.

En la figura 5.6 y 5.7, podemos observar los informes con los resultados finales para ser analizados por el equipo de negocio de la empresa.

	A	B	C	D
1	TOTAL_DESCUENTOS	BENEFICIO_OBTENIDO	PRODUCTO	RAZON_DESCUENTO
2	200.0	1800.0	POSTPAY_VODAFONE	EMPLEADOS
3	120.0	2280.0	POSTPAY_VODAFONE	WEB
4	810.0	1890.0	POSTPAY_VODAFONE	DTOPROM30
5	345.0	1955.0	POSTPAY_VODAFONE	SUPERCIERRE
6	160.0	1440.0	POSTPAY_VODAFONE	MICUMPLE
7	210.0	1890.0	POSTPAY_ORANGE	EMPLEADOS
8	435.0	2465.0	POSTPAY_ORANGE	SUPERCIERRE
9	450.0	1050.0	POSTPAY_ORANGE	DTOPROM30
10	75.0	1425.0	POSTPAY_ORANGE	WEB
11	190.0	1710.0	POSTPAY_ORANGE	MICUMPLE
12	750.0	1750.0	POSTPAY_YOIGO	DTOPROM30
13	180.0	1620.0	POSTPAY_YOIGO	MICUMPLE
14	160.0	1440.0	POSTPAY_YOIGO	EMPLEADOS
15	95.0	1805.0	POSTPAY_YOIGO	WEB
16	390.0	2210.0	POSTPAY_YOIGO	SUPERCIERRE
17	285.0	5415.0	PC/TABLET	WEB
18	2070.0	4830.0	PC/TABLET	DTOPROM30
19	450.0	4050.0	PC/TABLET	EMPLEADOS
20	1440.0	8160.0	PC/TABLET	SUPERCIERRE
21	390.0	3510.0	PC/TABLET	MICUMPLE
22	42.0	378.0	ACCESORIOS	EMPLEADOS
23	180.0	420.0	ACCESORIOS	DTOPROM30
24	52.0	468.0	ACCESORIOS	MICUMPLE
25	21.0	399.0	ACCESORIOS	WEB
26	78.0	442.0	ACCESORIOS	SUPERCIERRE
27	39.0	91.0	DUPLICADOS	DTOPROM30
28	9.5	85.5	DUPLICADOS	MICUMPLE
29	9.5	85.5	DUPLICADOS	EMPLEADOS
30	5.75	109.25	DUPLICADOS	WEB
31	18.75	106.25	DUPLICADOS	SUPERCIERRE
32	5220.0	12180.0	LIBRE	DTOPROM30
33	1890.0	10710.0	LIBRE	SUPERCIERRE
34	1500.0	13500.0	LIBRE	MICUMPLE
35	690.0	13110.0	LIBRE	WEB

Figura 5.6: Informe con beneficios y descuentos por producto

	A	B
1	VENTAS_POR_CIUDAD	CIUDAD
2	14	SANTA CRUZ DE TENERIFE
3	7	SANTIAGO DE COMPOSTELA
4	29	SAN SEBASTIAN
5	15	CIUDAD REAL
6	11	GUADALAJARA
7	10	TORRELAVEGA
8	11	CIUDADREAL
9	31	VALLADOLID
10	30	CASTELLÓN
11	21	SANTANDER
12	27	A CORUÑA
13	41	BARCELONA
14	12	CARTAGENA
15	7	PONTEVEDRA
16	18	SALAMANCA
17	13	ALICANTE
18	18	PAMPLONA
19	16	PALENCIA
20	11	ALMERÍA
21	55	VALENCIA
22	30	ZARAGOZA
23	21	CÁCERES
24	10	ALBACETE
25	19	VITORIA
26	7	CÓRDOBA
27	32	SEVILLA
28	21	MÁLAGA
29	21	GRANADA
30	26	BADAJOZ
31	12	SEGOVIA
32	13	OURENSE
33	11	LARIOJA
34	27	BILBAO
35	10	HUELVA

Figura 5.7: Informes con total de ventas por tienda en un día

Capítulo 6

CONCLUSIONES Y TRABAJO FUTURO

6.1. Conclusiones

Una de las tareas más complejas de nuestra implementación fue seleccionar cada una de las tecnologías y herramientas con las que íbamos a trabajar. Esto nos supuso una investigación sobre cada una de las opciones del mercado, basándonos en el grado de eficiencia, facilidad de implementación, compatibilidad, disponibilidad y escalabilidad que necesitábamos para nuestro proyecto, seleccionando siempre entornos gratuitos y de código libre que nos ofrecieran los recursos necesarios para desplegar nuestros programas. Esto nos supuso una gran cantidad de tiempo ya que al ser tecnologías desconocidas por nosotros, requerían comparaciones teóricas y prácticas entre sus alternativas.

Por otro lado, la mayor parte de la dificultad radicó en la configuración de los entornos y la realización de conexiones con los servidores remotos. Conseguir conectar todos los procesos de la arquitectura, siendo capaces de leer la misma estructura de datos y adaptándose a los lenguajes de programación correspondientes fue una de las tareas más complicadas por la cantidad de problemas de sintaxis, por un lado y adaptación al entorno por otro, con los que nos encontramos.

También cabe destacar que nuestro proyecto busca satisfacer las necesidades de un entorno empresarial. Esto requiere de una gran número de procesos de gran tamaño siendo ejecutados al mismo tiempo en equipos locales en vez de en potentes servidores con los que cuenta una gran empresa. Esto supuso un gran consumo de memoria RAM y ralentización de nuestros ordenadores a la hora de trabajar. A pesar de ello, hemos logrado construir una arquitectura compleja que se puede ejecutar a pequeña escala y permitiéndole ser escalable en un entorno de producción real.

Sobre los objetivos que nos planteamos para cubrir las necesidades de esta arquitectura completa, cabe destacar que se cumplieron la mayoría de ellos aunque encontramos alguno que no se ha podido completar.

Como objetivos cumplidos tenemos que comenzar indicando el ecosistema Big Data con una arquitectura Lambda desarrollado e implementada con éxito, obteniendo unos resultados con la estructura y forma deseados y basada en la composición de las capas

de *Real Time* y *Batch* totalmente independiente cada una de ellas y con sus respectivos métodos de acceso a las colas de mensajería, almacenamiento y proceso de análisis.

Se facilitó el análisis de los datos para que los departamentos de la empresa encargados de cumplir con los objetivos empresariales puedan cumplirlos. Esto se logró implementando herramientas de visualización en tiempo real donde pueden consultar los datos. Por otro lado generando informes útiles con la información necesaria para sacar las conclusiones debidas. Cabe destacar que teniendo esta arquitectura se puede utilizar para cualquier tipo de consultas necesarias.

Se consiguió desarrollar generadores de datos que simulan perfectamente el contenido de las tablas internas de la empresa y a su vez simulan el proceso de las ventas en base a eventos en tiempo real para posteriormente trabajar con estos datos y cumplir los objetivos de la empresa.

Por otro lado, también se desarrolló una Api Rest con métodos de escritura de estos datos generados sobre Azure Blob Storage para la capa *Batch*, consiguiendo emular de esta manera la inserción de los datos que realizará la compañía desde BigQuery.

Utilizamos colas de mensajería para el traspaso de eventos de información siendo muy óptimas para la capa *Real Time* aunque también se reutilizaron para la capa *Batch*.

Por parte de los repositorios de almacenamiento de la información procesada se cumple el objetivo de alojarlos en la nube con los servicios de Microsoft Azure que nos permite un año de prueba para proyectos de investigación y creando servidores SQL y Blob Storage.

Con respecto a la visualización de los resultados destacamos la implementación del dashboard de Grafana permitiendo al usuario la interacción con los propios datos seleccionando cuales desea analizar.

Como último objetivo cumplido se encuentra la creación de una segunda Api Rest con métodos de lectura y escritura sobre los datos analizados de Azure Blob Storage. Con esto permitimos el acceso de manera controlada a los resultados de la información facilitando así una posible nuevas funcionalidades de explotación sobre estos datos para una ampliación de la arquitectura.

Por la parte de los objetivos no cumplidos cabe destacar la falta del data lake. Para nuestro proyecto, MongoDB iba a ser nuestro repositorio no relacional del histórico de datos de la empresa, como indicamos en la parte de implementación y que es una parte imprescindible en una arquitectura Big Data. Esto se debe al abandono del integrante del grupo encargado de esta sección viéndonos obligados a prescindir de ello y realizar un cambio de alcance en el proyecto, integrando este repositorio como bases de datos relacionales en uno de los servidores de Azure de los que empleamos en otra parte de la arquitectura.

6.2. Trabajo futuro

Esta arquitectura ofrece una serie de oportunidades para ser ampliada y mejorada en distintos aspectos:

- Integración con entorno de facturación: A mayores, se ha planteado aprovechar nuestra arquitectura para que el departamento de IT de la empresa realice una integración con la plataforma de facturación de la empresa. La idea de dicha integración es mejorar los tiempos de la información procesada en tiempo real y evitar tener que realizar el volcado a las bases de datos para luego ser consultada. Se quiere construir una vista en la que el personal de tienda pueda canjear promociones particulares a clientes, enviando esta información directamente a nuestras colas de mensajería y pasando finalmente por cada una de las etapas desarrolladas.
- Data Science: A nivel de análisis de datos ofrece gran margen para realizar ampliaciones en la consulta de datos específicos y KPIS que representen una mejora en la toma de decisiones, en el desarrollo y el crecimiento de la empresa. Esto representa un impacto grande en la digitalización de la compañía, siendo esto un punto clave para competir contra grandes empresas de la industria de la tecnología.
- Machine learning: Inclusión de un algoritmo de inteligencia artificial que pueda adoptar sabiduría acerca de la experiencia del éxito de las promociones aplicadas a los clientes en base a los objetivos de la compañía. Siendo el mismo algoritmo el que realice nuevas campañas particulares a lanzar de cara a objetivos planteados.
- Despliegue en entornos de producción: Configurar “clusters” en la nube que puedan ejecutar los programas de procesamiento que puedan realizar estos procesos de manera automática sin necesidad de ejecutar en un entorno local.

  Habilitar “dockers” para alojar las aplicaciones de colas de mensajería y poder utilizarlas de manera remota.
- Mejoras: Por un ajuste de alcance decidimos optar por la utilización de un repositorio SQL que bien puede ser sustituido por bases de datos NoSQL. Estas nos permiten una arquitectura distribuida en la que podemos almacenar nuestros datos. También son más óptimas a la hora de trabajar con grandes cantidades de datos.

Capítulo 7

CONCLUSIONS AND FUTURE WORK

7.1. Conclusions

One of the most complex tasks of our implementation was to select each of the technologies and tools we were going to work with. This involved researching each of the options on the market, based on the degree of efficiency, ease of implementation, compatibility, availability and scalability that we needed for our project, always selecting free and open source environments that offered us the necessary resources to deploy our programmes. This meant a great deal of time for us, as the technologies were unknown to us, requiring theoretical and practical comparisons between their alternatives.

On the other hand, most of the difficulty lay in configuring the environments and making connections with the remote servers. Connecting all the processes of the architecture, being able to read the same data structure and adapting to the corresponding programming languages was one of the most complicated tasks due to the number of syntax problems, on the one hand, and adaptation to the environment, on the other.

It should also be noted that our project aims to meet the needs of a business environment. This requires a large number of large processes being executed at the same time on local computers instead of on powerful servers that a large company has. This meant a large consumption of RAM memory and slowing down our computers when working. Despite this, we have managed to build a complex architecture that can be run on a small scale and allows it to be scalable in a real production environment.

Regarding the objectives we set ourselves to cover the needs of this complete architecture, it should be noted that most of them were met, although there were some that could not be completed.

As fulfilled objectives we have to start by indicating the Big Data ecosystem with a Lambda architecture developed and successfully implemented, obtaining results with the desired structure and form and based on the composition of the layers of "Real Time" and "Batch", each of them totally independent and with their respective methods of access to the messaging queues, storage and analysis process.

The analysis of the data was made easier for the company's departments in charge of

meeting business objectives. This was achieved by implementing real-time visualisation tools where they can query the data. On the other hand, by generating useful reports with the necessary information to draw the right conclusions. It should be noted that having this architecture can be used for any type of queries needed.

It was possible to develop data generators that perfectly simulate the content of the company's internal tables and at the same time simulate the sales process based on real time events in order to subsequently work with this data and fulfil the company's objectives.

On the other hand, a Rest Api was also developed with methods for writing this data generated on Azure Blob Storage for the Batch layer, thus emulating the insertion of the data that the company will carry out from BigQuery.

We used messaging queues for the transfer of information events, being very optimal for the Real Time layer, although they were also reused for the Batch layer.

As for the storage repositories of the processed information, the objective of hosting them in the cloud with Microsoft Azure services is met, which allows us a year of testing for research projects and creating SQL servers and Blob Storage.

With regard to the visualisation of the results, we highlight the implementation of the Grafana dashboard, allowing the user to interact with the data themselves by selecting which data they wish to analyse.

The last objective achieved is the creation of a second Api Rest with read and write methods on the data analysed from Azure Blob Storage. With this we allow controlled access to the results of the information, thus facilitating possible new functionalities of exploitation on this data for an extension of the architecture.

On the part of the objectives not met, it is worth highlighting the lack of a data lake. For our project, MongoDB was going to be our non-relational repository of the company's historical data, as we indicated in the implementation section and which is an essential part of a Big Data architecture. This is due to the departure of the member of the group in charge of this section, forcing us to dispense with it and make a change of scope in the project, integrating this repository as a relational database in one of the Azure servers that we use in another part of the architecture.

7.2. Future work

This architecture offers a number of opportunities to be extended and improved in different aspects:

- Integration with invoicing environment: In addition, it has been proposed to take advantage of our architecture so that the company's IT department can integrate with the company's invoicing platform. The idea of this integration is to improve the times of the information processed in real time and avoid having to dump it to the databases to be consulted later. We want to build a view in which the shop staff can redeem particular promotions to customers, sending this information directly to our messaging queues and finally passing through each of the developed stages.

- Data sience: At the data analysis level it offers great scope for extensions to query specific data and KPIS that represent an improvement in decision making, business development and growth. This represents a major impact on the digitisation of the company, which is a key point to compete against large companies in the technology industry.
- Machine learning: Inclusion of an artificial intelligence algorithm that can adopt wisdom about the experience of the success of promotions applied to customers based on the objectives of the company. Being the same algorithm the one that carries out new particular campaigns to be launched in order to meet the objectives set.
- Deployment in production environments: Configure 'clusters' in the cloud that can run processing programs that can perform these processes automatically without the need to run in a local environment.

  Enable dockers to host messaging queuing applications for remote use.
- Improvements: For a scope adjustment we decided to opt for the use of a SQL repository that can be replaced by NoSQL databases. These allow us a distributed architecture in which we can store our data. They are also more optimal when working with large amounts of data.

Capítulo 8

CONTRIBUCIONES

La distribución y la contribución del proyecto entre ambos integrantes se describe con los puntos principales del proyecto donde hubo una participación por ambas partes:

- Implementación de la arquitectura
- Redacción de la Memoria

A continuación, cada participante realizó:

8.1. Gabriel Emilio Lugo Estévez

8.1.1. Investigación

- Encargado de la investigación de la programación funcional con Scala para el desarrollo de programas en Big Data.
- Investigación y comparación de las metodologías de extracción de datos en entornos Big Data.
- Investigación y comparación de las tecnologías de procesamiento de datos en tiempo real.
- Investigación y comparación de las tecnologías de procesamiento de datos en modo *Batch*.

8.1.2. Prototipos

- Aplicación en Python para la extracción de datos de la *API* de Twitter.
- Aplicación en Python para la extracción de datos de BigQuery.
- Aplicación en Java para pruebas de concepto de productores y consumidores de kafka.
- Aplicación en Python para pruebas de concepto de productores y consumidores de kafka.
- Generador de datos en ficheros *JSON*.

- Generador de datos en ficheros *CSV*.
- Desarrollo de un *API Rest* con Sring Boot para pruebas de concepto.
- Aplicación en Java para pruebas de concepto de peticiones al *API*
- Aplicación en Scala para probar el procesamiento *Batch* con *Databricks*

8.1.3. Implementación en la arquitectura

- Generador de datos de clientes con su productor Kafka
- Generador de datos de ventas con su productor Kafka
- *API* de acceso a los datos en Azure Storage
- Aplicación consumidora de datos de Kafka desarrollado en Java
- Programa de volcado de datos a través de *API*
- Aplicación en Scala Spark que lee los datos del repositorio Cloud y los procesa de manera distribuida.
- Programa que explota los datos generando informes a partir del procesamiento distribuido.

8.1.4. Memoria

Apartados redactados en el capítulo “Introducción”:

- Introducción general del proyecto.
- Motivaciones de realización para llevar a cabo este trabajo.
- Objetivos generales planteados para la realización del trabajo en colaboración con Álvaro Fuentes Solas.
- Metodología y el plan de trabajo utilizado para el desarrollo del proyecto

Apartados redactados en el capítulo “Estado del Árte”:

- Metodologías de extracción de datos en entornos Big Data. Con sus respectivas comparaciones y justificación de la metodología elegida para nuestro proyecto.
- Tecnologías de procesamiento de los datos en tiempo real, comparando cada una de las alternativas, demostrando ejemplos, ventajas y desventajas. Justifico la elección de la tecnología utilizada en la implementación.

Apartados redactados en el capítulo “Implementación”:

- Resumen del capítulo.
- Apartado “Diseño y estructura”
- Diagrama arquitectura Lambda implementada en nuestro proyecto.
- Resumen del apartado “Software implementado”.

- Explicación del funcionamiento de la extracción de los datos en nuestro proyecto, explicando los prototipos desarrollados para pruebas de concepto.
- Explico en detalle el funcionamiento de los generadores de datos desarrollados.
- Funcionamiento de las colas de mensajería y su implementación en nuestro proyecto. Explicación de los comandos necesarios para inicar Kafka en entornos locales.
- Funcionamiento de la capa *Batch* implementada en nuestro proyecto, esta incluye, el consumo de datos, el volcado mediante de las *APIs*, su lectura de manera distribuida, el procesamiento de los datos y su posterior almacenamiento. Para concluir explico la explotación de la información en modo *Batch*.

Apartados redactados en el capítulo "Resultados":

- Redacción de los resultados obtenidos con el procesamiento batch.
- Demostración de informes con los resultados obtenidos, para los análisis de negocio.

Apartados redactados en el capítulo "Conclusiones":

- Conclusiones recabadas de nuestro proyecto, en colaboración con Álvaro Fuentes Solas
- Explicación de los objetivos cumplidos y los no cumplidos con respecto a la parte de mi implementación, justificando las razones.
- Trabajos futuros y los ámbitos aplicables para evolucionar la arquitectura implementada.

8.1.5. Aportes extras

- Encargado junto al departamento de la empresa para el cierre del ámbito aplicable del proyecto para solucionar una posible mejora en la empresa. A su vez encargado de la comunicación directa con el equipo interno de la empresa para llevar a cabo las reuniones necesarias.
- Realización de la construcción del diseño de la arquitectura completa a implementar con sus debidos flujos y procesos.
- Implementación de una guía práctica de programación funcional con Scala para los integrantes del equipo. Realización de guía para el uso de las colas Apache Kafka en los entornos locales.

8.2. Álvaro Fuentes Solas

8.2.1. Investigación

- Investigación y comparación de las tecnologías de procesamiento de datos en modo *Real Time*.
- Investigación sobre los distintos tipos de acceso a las bases de datos Cloud.
- Investigación para reducir los costes de inserción de datos en *streaming* sobre las bases de datos

8.2.2. Prototipos

- Aplicación en Java para la inserción de datos en SQL Server de Azure desde ficheros *JSON* y CSV.
- Lectura de datos en Python desde Apache Kafka y su inserción en ficheros *JSON* y *CSV*.
- Aplicación en Python para la inserción de datos en SQL Server de Azure desde ficheros *JSON* y *CSV*.
- Lectura de datos de SQL Server de Azure mediante los accesos de ODBC y JDBC en Java.
- Lectura de datos de SQL Server de Azure mediante los accesos de ODBC y JDBC en Python.

8.2.3. Implementación en la arquitectura

- Establecimiento de la cuenta de Microsoft Azure.
- Creación de la base de datos SQL en el servidor Cloud.
- Lectura de los datos desde Apache Kafka en "streaming".
- Implementación de los procesos de análisis de datos en *streaming* con PySpark.
- Volcado de datos en Python de los datos analizados a tablas SQL del servidor Cloud.
- Volcado de datos en Python de los datos sin analizar a las tablas SQL del servidor Cloud como sustituto del repositorio noSQL.
- Creación y diseño del *dashboard* junto con la preparación para una posible interacción con él.

8.2.4. Memoria

Apartados redactados en el capítulo "Introducción":

- Objetivos generales planteados para la realización del trabajo en colaboración con Gabriel Emilio Lugo.
- Explico la estructura en la que se ha dividido.

Apartados redactados en el capítulo "Estado del Arte":

- Introducción y descripción del significado de Big Data y lo que este término representa
- Desarrollo y explicación de los tipos de arquitecturas Big Data que nos podemos encontrar
- Comparación de las distintas herramientas de mensajería de los datos, distinguiendo los puntos fuertes de cada una de ellas y que situaciones son los óptimos para su uso.

- Explicación de los tipos de almacenamiento en el sector del Big Data junto con los beneficios que ofrecen según la estructura de los datos y su análisis.
- Desarrollo de los *softwares* de visualización de datos más demandados y que finalidades tienen cada uno de ellos.
- Introducción al Cloud Computing, porque es una parte tan importante del Big Data y cuáles son algunos de sus inconvenientes y retos futuros.

Apartados redactados en el capítulo "Implementación":

- Redacto la parte del "Diseño y Estructura" de nuestro arquitectura Lambda correspondiente a la capa de *Real Time*.
- Explicación detalla de cada proceso correspondiente a la capa de *Real Time*.

Apartados redactados en el capítulo "Resultados":

- Explicación de la creación de un *dashboard* para visualización de los datos en Grafana.
- Descripción de los pasos para la conexión del *dashboard* a una base de datos.
- Desarrollo de los resultados visuales con los datos trabajados.

Apartados redactados en el capítulo "Conclusiones":

- Conclusiones recabadas de nuestro proyecto, en colaboración con Gabriel Emilio Lugo.
- Explicación de los objetivos cumplidos y los no cumplidos con respecto a la parte de la capa *Real Time*, justificando las razones.

Bibliografía

V. Abramova and J. Bernardino. Nosql databases: Mongodb vs cassandra. In *Proceedings of the international C* conference on computer science and software engineering*, pages 14–22, 2013.

AbsentData. Microsoft power bi pros and cons. https://www.absentdata.com/power-bi-pros-and -cons/, 2020.

G. Ads. Archivo csv: definición. https://support.google.com/google-ads/answer/9004364?hl=es.

S. M. Ali, N. Gupta, G. K. Nayak, and R. K. Lenka. Big data visualization: Tools and challenges. In *2016 2nd International Conference on Contemporary Computing and Informatics (IC3I)*, pages 656–660. IEEE, 2016.

M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. Katz, A. Konwinski, G. Lee, D. Patterson, A. Rabkin, I. Stoica, et al. A view of cloud computing. *Communications of the ACM*, 53(4):50–58, 2010.

M. Armbrust, T. Das, J. Torres, B. Yavuz, S. Zhu, R. Xin, A. Ghodsi, I. Stoica, and M. Zaharia. Structured streaming: A declarative api for real-time applications in apache spark. In *Proceedings of the 2018 International Conference on Management of Data*, pages 601–613, 2018.

M. Azure. Cómo usar el iniciador de spring boot para azure storage. https://docs.microsoft.com/es-es/azure/developer/java/spring-framework/configure-spring-boot-starter-java-app-with-azure-storage/, 2021.

M. N. Birje, P. S. Challagidad, R. Goudar, and M. T. Tapale. Cloud computing review: concepts, technology, challenges and security. *International Journal of Cloud Computing*, 6(1):32–57, 2017.

E. Carlos Espilez y Rubén Villa. Arquitecturas lambda sobre azure. https://informatica.ucm.es/vii-semana-de-la-informatica-2021everis, 2021.

G. Cloud. Bigquery api. https://cloud.google.com/bigquery/docs/reference/rest, 2020.

I. Confluent. Kafka vs. pulsar vs. rabbitmq: Performance, architecture, and features compared. https://www.confluent.io/kafka-vs-pulsar/, 2014-2021.

T. da Silva Morais. Survey on frameworks for distributed computing: Hadoop, spark and storm. In *Proceedings of the 10th Doctoral Symposium in Informatics Engineering-DSIE*, volume 15, 2015.

P. Dobbelaere and K. S. Esmaili. Kafka versus rabbitmq: A comparative study of two

industry reference publish/subscribe implementations: Industry paper. In *Proceedings of the 11th ACM international conference on distributed and event-based systems*, pages 227–238, 2017.

C. Fehling, F. Leymann, R. Retter, W. Schupeck, and P. Arbitter. *Cloud computing patterns: fundamentals to design, build, and manage cloud applications*. Springer, 2014.

V. V. Fernández. Spark vs hadoop, ¿cuál es mejor? https://www.esic.edu/rethink/tecnologia/spark-vs-hadoop-cual-es-mejor, 2019.

A. S. Foundation. Home - hadoop - apache software foundation. https://cwiki.apache.org/confluence/display/hadoop.

A. S. Foundation. Apache storm. https://storm.apache.org/about/simple-api.html, 2019a.

A. S. Foundation. Apache storm. https://storm.apache.org/about/integrates.html, 2019b.

A. S. Foundation. Apache storm. https://storm.apache.org/about/scalable.html, 2019c.

T. A. S. Foundation. Apache flink: What is apache flink? — architecture. https://flink.apache.org/flink-architecture.html, 2014-2021.

T. A. S. Foundation. Apache spark™ - unified analytics engine for big data. https://spark.apache.org/, 2018a.

T. A. S. Foundation. Spark streaming | apache spark. https://spark.apache.org/streaming/, 2018b.

T. A. S. Foundation. Spark sql dataframes | apache spark. https://spark.apache.org/sql/, 2018c.

B. Furht. Cloud computing fundamentals. In *Handbook of cloud computing*, pages 3–19. Springer, 2010.

G. Garrison, S. Kim, and R. L. Wakefield. Success factors for deploying cloud computing. *Communications of the ACM*, 55(9):62–68, 2012.

C. Győrödi, R. Győrödi, G. Pecherle, and A. Olah. A comparative study: Mongodb vs. mysql. In *2015 13th International Conference on Engineering of Modern Electric Systems (EMES)*, pages 1–6. IEEE, 2015.

Z. Hasani, M. Kon-Popovska, and G. Velinov. Lambda architecture for real time big data analytic. *ICT Innovations*, pages 133–143, 2014.

. B. N. Hausenblas, M. Lambda architecture. http://lambda-architecture.net/, 2015.

B. Hayes. Cloud computing, 2008.

S. Hussain. 4 big data architectures, data streaming, lambda architecture, kappa architecture, and unifield architecture. https://medium.com/dataprophet/4-big-data-architectures-data-streaming -lambda-architecture-kappa-architecture-and-unifield-d9bcbf711eb9, 2018.

Keepler. Análisis de monitorización de servicios en la nube con grafana. https://keepler.io/2019/11/analisis-de-monitorizacion-de-servicios-en-la-nube-con-grafana/.

D. Keim, H. Qu, and K.-L. Ma. Big-data visualization. *IEEE Computer Graphics and Applications*, 33(4):20–21, 2013.

D. Klein, P. Tran-Gia, and M. Hartmann. Big data. *Informatik-Spektrum*, 36(3):319–323, 2013.

R. Kumar and S. Charu. An importance of using virtualization technology in cloud computing. *Global Journal of Computers & Technology*, 1(2), 2015.

A. Lakshman and P. Malik. Cassandra: a decentralized structured storage system. *ACM SIGOPS Operating Systems Review*, 44(2):35–40, 2010.

J. Lennon. Introduction to json. In *Beginning couchdb*, pages 87–105. Springer, 2009.

L. Li. 16 - the future of academic libraries in the digital age. In D. Baker and W. Evans, editors, *Trends, Discovery, and People in the Digital Age*, Chandos Digital Information Review, pages 253–268. Chandos Publishing, 2013. ISBN 978-1-84334-723-1. doi: https://doi.org/10.1016/B978-1-84334-723-1.50016-4. URL https://www.sciencedirect.com/science/article/pii/B9781843347231500164.

A. Oussous, F.-Z. Benjelloun, A. A. Lahcen, and S. Belfkih. Big data technologies: A survey. *Journal of King Saud University-Computer and Information Sciences*, 30(4): 431–448, 2018.

C. Ozgur, M. Kleckner, and Y. Li. Selection of statistical software for solving big data problems: A guide for businesses, students, and universities. *SAGE Open*, 5 (2):2158244015584379, 2015.

A. Paredes. 5 problemas comunes en el análisis de datos y como solucionarlos. https://intanis.com/5-problemas-analisis-de-datos-y-como-solucionarlos/, 2020. Accessed: 2020-21-06.

M. Pathirage. kappa-architecture.com. http://milinda.pathirage.org/kappa-architecture.com/, 2017.

P. Pedamkar. Big data architecture. https://www.educba.com/big-data-architecture/?source=leftnav.

A. Rabiee. Analyzing parameter sets for apache kafka and rabbitmq on a cloud platform, 2018.

I. Red Hat. ¿quÉ es una api? https://www.redhat.com/es/topics/api/what-are-application-programming-interfaces, 2021.

J. R. Revision. Tweepy. https://www.tweepy.org/, 2009,2020.

E. B. . M. SCHOOL. Apache spark: Introducción, qué es y cómo funciona. https://www.esic.edu/rethink/tecnologia/apache-spark-introduccion-que-es-y-como-funciona, 2018.

O. O. D. Science. Advantages and best uses of four popular data visualization

tools. https://medium.com/@ODSC/advantages-and-best-uses-of-four-popular-data-visualization-tools-63df88619662, Oct 17, 2018.

. I. E. S.L.U. Redis: otro concepto de base de datos. https://www.ionos.es/digitalguide/hosting/ cuestiones-tecnicas/que-es-redis/, 2021.

M. Tascón. Introducción: Big data. pasado, presente y futuro. *Telos: Cuadernos de comunicación e innovación*, (95):47–50, 2013.

I. TWITTER. Twitter api. https://developer.twitter.com/en/docs/twitter-api, 2021.

B. Varghese and R. Buyya. Next generation cloud computing: New trends and research directions. *Future Generation Computer Systems*, 79:849–861, 2018.

I. VMware. Spring initializer. https://start.spring.io, 2013-2021.

W. Xiao and J. Hu. Sweclat: a frequent itemset mining algorithm over streaming data using spark streaming. *The Journal of Supercomputing*, 76(10):7619–7634, 2020.

PASCAL
ENERO 2018
Ult. actualización 15 de junio de 2021
LaTeX lic. LPPL & powered by TEFLON CC-ZERO